Three Sigma Labs

Code Audit

MAPLE

Maple Finance Lending Protocol

Disclaimer

Code Audit

Maple Finance 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 Finance Lending Protocol

Table of Contents

Summary

Code Audit

Maple Finance Lending Protocol

Summary

Three Sigma Labs audited Maple Finance's V2 Private smart contracts in a 6 person week engagement. The audit was conducted from 10-04-2023 to 21-04–2023.

Protocol Description

Maple Finance is an institutional crypto-capital network built on Ethereum. Maple 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. Built with both traditional financial institutions and decentralized finance leaders, Maple is transforming capital markets by combining industry-standard compliance and due diligence with the frictionless lending enabled by smart contracts and blockchain technology. Maple is the gateway to growth for financial institutions, pool delegates and companies seeking capital on-chain.

Scope

Code Audit

Maple Finance Lending Protocol

Scope

The audit examined Maple Finance's V2 Private core contracts.

V2 Private Protocol Contracts

mapleLabs/fixed-term-loan-private/contracts		
Contract	**Scope**	**Mainnet Action**
MapleLoan.sol	Logic and styling changes	Upgrade
MapleLoanInitializer.sol	Logic changes	New deployment
MapleLoanV5Migrator.sol	Logic changes	New deployment
Refinancer.sol	Styling changes	New deployment
mapleLabs/fixed-term-loan-manager-private/contracts		
LoanManager.sol	Logic and storage changes	Upgrade
LoanManagerInitializer.sol	Logic changes	New deployment
mapleLabs/globals-v2-private/contracts		
MapleGlobals.sol	Logic and storage changes	Upgrade
mapleLabs/open-term-loan-private/contracts		
MapleLoan.sol	New contract	New deployment
MapleLoanFactory.sol	New contract	New deployment
MapleLoanInitializer.sol	New contract	New deployment
MapleRefinancer.sol	New contract	New deployment
mapleLabs/open-term-loan-manager-private/contracts		
LoanManager.sol	New contract	New deployment

LoanManagerFactory.sol	New contract	New deployment
LoanManagerInitializer.sol	New contract	New deployment
mapleLabs/pool-v2-private/contracts		
Pool.sol	Styling changes, same bytecode	New deployment
PoolDeployer.sol	Logic changes	New deployment
PoolManager.sol	Logic changes	Upgrade
PoolManagerInitializer.sol	Styling changes, same bytecode	New deployment

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 8f1b5ad15b9728cd79b3756cab18d0926ae9480c on the maple-labs/maple-core-v2-private 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 Finance 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 difficulty of the exploit. The following table summarizes the general expected classification according to severity and difficulty; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

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

Project Dashboard

Code Audit

Maple Finance Lending Protocol

Project Dashboard

Application Summary

Name	Maple Finance
Commit	8f1b5ad1
Language	Solidity
Platform	Ethereum

Engagement Summary

Timeline	10 Abril to 21 April, 2023
Nº of Auditors	3
Review Time	6 person weeks

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	0	0	0
High	3	3	0
Medium	9	5	4
Low	9	3	6
None	14	9	5

Category Breakdown

Suggestion	11
Optimization	10
Bug	13
Access Control	1

Code Maturity Evaluation

Code Audit

Maple Finance 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 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. Loans can be DoSed by frontrunning fund transactions with return funds [3S-MAPLE-M02, M-06].
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.

Findings

Code Audit

Maple Finance Lending Protocol

Findings

3S-MAPLE-H01

Fixed term loans can be deployed with a wrong fee manager and possibly steal all funds.

Id	3S-MAPLE-H01
Classification	High
Severity	Critical
Difficulty	High
Category	Bug
Relevant Links	fixed-term-loan-private/MapleLoanInitializer#L127 globals-v2-private/MapleGlobals#L75 globals-v2-private/MapleGlobas#L228

Description

In `MapleLoanInitializer`, the address of the fee manager is not whitelisted.

Exploit scenario

- Borrower agrees with delegate on the terms and creates a loan with the correct fee manager;
- Attacker sees the transaction and creates a loan with the same arguments, changing only the fee manager address. They can even choose a salt that makes the first bytes of the malicious loan contract equal to the valid loan contract;

- The pool delegate sees the malicious loan address and funds it thinking it is the valid one;
- All funds are sent to the attacker.

Recommendation

Whitelist the Fee Manager by adding an `isInstanceOf` check in `MapleGlobals`.

Status

Addressed in the following PR:

https://github.com/maple-labs/fixed-term-loan-private/pull/282

3S-MAPLE-H02

Pool Delegates can set a really high origination fee and steal all pool funds.

Id	3S-MAPLE-H02
Classification	High
Severity	Critical
Difficulty	High
Category	Bug
Relevant Links	fixed-term-loan-private/MapleLoanInitializer#L146 fixed-term-loan-private/MapleLoan#L351

Description

The delegate origination fee can be set to any value at any time and anyone can deploy a loan, so the delegate can receive all the funds of a loan.

Exploit scenario

- gather funds to a pool with a regular origination fee
- given enough funds, increase the origination fee to the pool amount minus the platformOriginationFee (which can be close to zero if the delegate creates the loan with 1 payment interval and 1 number of payments)
- create a *fake* loan (anyone can deploy a loan) and fund it

---

Recommendation

The following mitigation options were proposed:

- Redeploy factory and fix anyone can deploy a loan;
- Set an origination fee limit;
- Don’t allow changing the origination fee;
- Set a timelock/minimum withdrawal cycles to change the origination fee;
- Require governor approval on fee change.

Status

Addressed in the following PR:

https://github.com/maple-labs/fixed-term-loan-private/pull/286

3S-MAPLE-H03

Pool Delegates can steal all the pool's funds by setting a malicious Withdrawal Manager.

Id	3S-MAPLE-H03
Classification	High
Severity	Critical
Difficulty	High
Category	Bug
Relevant Links	pool-v2-private/PoolManager#L207

Description

The current `PoolManager` contract allows the pool manager to change the `withdrawalManager` contract without any checks using the `setWithdrawalManager()` function. Since the withdrawal manager is responsible for holding the redeemable tokens and informing the pool which tokens can be redeemed, corrupting this contract gives complete control of the pool's funds to the pool delegate.

Exploit scenario

- Pool delegate deposits one token into the Pool to get a single share
- Pool delegate sets a malicious withdrawal manager, allowing them to redeem a single share for the entire pool balance
- Pool delegate redeems their share and receives all the funds in the pool

---

Recommendation

Remove the `setWithdrawalManager()` function or pause it by default.

Status

Addressed in the following PR:

https://github.com/maple-labs/pool-v2-private/pull/272

3S-MAPLE-M01

Open term loans can be created with zero platform service fee if borrowers create them right after a pool has been deployed.

Id	3S-MAPLE-M01
Classification	Medium
Severity	Medium
Difficulty	Medium
Category	Bug
Relevant Links	open-term-loan-private/MapleLoanInitializer#L99

Description

The platform service fee is set on loan deployment; however, when a pool is deployed, its platform service fee is 0 (has not been set yet), so a borrower could deploy a loan right after a pool deployment and have 0 platform service fee.

Recommendation

Set the `platformServiceFee` when a loan is funded instead of when created, similarly to what happens on the fixed term loan.

Status

Acknowledged by the team.

3S-MAPLE-M02

`fundLoan()` can be DoSed if `returnFunds()` is called before it.

Id	3S-MAPLE-M02
Classification	Medium
Severity	Medium
Difficulty	Low
Category	Suggestion
Relevant Links	fixed-term-loan-private/MapleLoan#L237 fixed-term-loan-private/MapleLoan#L353 withdrawal-manager-private/WithdrawalManager#L337

Description

It's possible to stop loans from being funded by calling `returnFunds()`.

Exploit scenario

- attacker sees `fundLoan(...)` transaction and calls `returnFunds(...)` with an arbitrary amount
- `returnFunds(...)` increases `_drawableFunds`
- `fundLoan(...)` overrides `_drawableFunds`, which means that the amount sent earlier will be unaccounted
- the require that there are 0 unaccounted funds fails and the transaction reverts

The lenders have incentive to do this if they want to withdraw and a loan would remove liquidity, see `getRedeemableAmounts(...)`.

Recommendation

In `fundLoan(...)`, increase `_drawableFunds` instead of setting it.

Status

Addressed in the following PR:

https://github.com/maple-labs/fixed-term-loan-private/pull/285

3S-MAPLE-M03

A pool delegate should not be allowed to be a borrower

Id	3S-MAPLE-M03
Classification	Medium
Severity	Medium
Difficulty	High
Category	Access Control
Relevant links	globals-v2-private/MapleGlobals#L214-L219 globals-v2-private/MapleGlobals#L242-L252

Description

A Pool Delegate should not be allowed to be a borrower otherwise they can leave with the funds at any time with no questions asked. We believe this is not the intended behavior, but currently the code allows it to happen (which means it could happen by mistake, for example by copy pasting the wrong address).

Recommendation

Suggestions to mitigate this issue (to implement in maple globals):

- In `setValidBorrower`, check if the address is a pool delegate.
- In `setValidPoolDelegate`, check if the address is a borrower.

Status

Acknowledged by the team.

3S-MAPLE-M04

Pool Delegates can set an unreasonably large delegate management fee rate at anytime

Id	3S-MAPLE-M04
Classification	Medium
Severity	High
Difficulty	High
Category	Bug
Relevant Links	pool-v2-private/PoolManager#L172-L177

Description

Using the function `setDelegateManagementFeeRate()` the pool manager can set an unreasonably large `delegateManagementFeeRate` (all the way up to 100%) at any time, stopping payments from reaching the pool (i.e. the liquidity providers).

Since the loan manager updates the `delegateManagementFeeRate` every time a payment is made (and a new is added), this change will have an almost immediate impact, and the liquidity providers will have no choice but to accept these new terms. Some of them could try to withdraw their funds, but the majority of them would not be allowed to do so since the loan had already been made and the funds were locked.

A pool delegate could therefore start by setting a low management fee to attract LPs to his pool and, after loans had been issued, this fee could be drastically increased without any governor or LPs permission, to the detriment of the LPs.

Recommendation

Allow the governor to limit this value or give the governor permission to accept or reject a change in the `delegateManagementFeeRate`. Alternatively, `delegateManagementFeeRate` could be set in deployment and not allowed to change, or at least stay constant on a per loan basis.

Status

Acknowledged by the team.

3S-MAPLE-M05

`MapleGlobals`, `activatePoolManager()` has no check that the pool manager is actually a valid pool manager.

Id	3S-MAPLE-M05
Classification	Medium
Severity	Medium
Difficulty	High
Category	Bug
Relevant Links	globals-v2-private/MapleGlobals#L109 pool-v2-private/PoolManager#115 pool-v2-private/PoolManager#126

Description

In `MapleGlobals`, `activatePoolManager()`, there is no check that the Pool Manager and Pool Delegate are valid, so the governor can mistakenly set malicious addresses as activated.

Exploit scenario

- governor was tricked into activating a fake pool manager, which works because it does not validate if the pool delegate of the pool manager is a valid pool delegate nor if the poolManager is a valid poolManager
- a real pool delegate calls `acceptPoolDelegate()` on a pool manager, but the fakePoolManager frontruns it and calls `transferOwnedPoolManager(...)` with the `toDelegate` argument as the pool delegate

- the pool delegate is now the owner of the fake pool in the mapping `poolDelegates[delegate_].ownedPoolManager = poolManager_`
- the `acceptPoolDelegate` transaction to the real pool manager will fail because the pool delegate is already the owner of the fake pool manager `(require(poolDelegates[delegate_].ownedPoolManager == address(0), "MG:APM:ALREADY_OWNS");` in `activatePoolManager`
- the pool delegate does not notice that he is the owner of the fake pool manager instead of the real pool manager and calls deposit cover, which has malicious code, and sends the funds to the attacker

Recommendation

Check the validity of the Pool Delegate and Pool Manager in `activatePoolManager(...)`.

Status

Addressed in the following PR:

https://github.com/maple-labs/globals-v2-private/pull/62

3S-MAPLE-M06

DoS attack if assets are transferred before a `fundLoan()` call.

Id	3S-MAPLE-M06
Classification	Medium
Severity	Medium
Difficulty	Low
Category	Bug
Relevant Links	[fixed-term-loan-private/MapleLoan#L334]()

Description

In fixed term loans, a DoS attack is possible if assets are transferred into the `MapleLoan` before the loan is funded.

Exploit scenario

- delegate funds loan
- attacker frontruns delegate and transfers `fundsAsset` to the `MapleLoan`
- funding reverts due to the `require` that the unaccounted funds are 0

Recommendation

Skim before funding.

Status

Addressed in the following PR:

https://github.com/maple-labs/fixed-term-loan-manager-private/pull/34

3S-MAPLE-M07

Liquidation can be finished without calling `triggerDefault()` (and repossessing the loan) if pool delegate or governor call `finishCollateralLiquidation(...)` after impairment.

Id	3S-MAPLE-M07
Classification	Medium
Severity	High
Difficulty	High
Category	Bug
Relevant Links	fixed-term-loan-manager-private/LoanManager#L288 fixed-term-loan-manager-private/LoanManager#L376 pool-v2-private/PoolManager#L260

Description

The usual flow is to impair a loan and then call `triggerDefault()` followed by `finishCollateralLiquidation()`; however, it's possible to call `finishCollateralLiquidation()` without calling `triggerDefault()`, leading to the collateral in the loan not being liquidated and the cover of the delegate being used.

Exploit scenario

- delegate or governor impair a loan

- delegate or governor call `finishCollateralLiquidation(...)` without calling `triggerDefault(...)` first
- The `liquidationInfo` is deleted, the collateral in the loan is not repossessed and the cover of the delegate is used to cover the remaining losses (if available)
- `principalOut` in the loan manager will decrease by the principal amount
- The `paymentId` of the loan will not be deleted, so if the borrower decides to make a payment / close the loan at any time, and this call updates the `principalOut` the following 2 scenarios are possible:
  - If the loan is the only loan in the loan manager, it will revert
  - If there are more loans, it will decrease the `principalOut` and it can make the other loans' payments fail

Recommendation

Check if the address of the liquidator is 0 at the beginning of the `finishCollateralLiquidation()` function.

Status

Addressed in the following PR:

https://github.com/maple-labs/fixed-term-loan-manager-private/pull/33

3S-MAPLE-M08

Pool Delegates can accidentally lock out of their funds LPs in the middle of redeeming.

Id	3S-MAPLE-M08
Classification	Medium
Severity	Medium
Difficulty	High
Category	Bug
Relevant Links	pool-v2-private/PoolManager#L207

Description

Pool delegates are allowed to change the withdrawal manager address at any time using the `setWithdrawalManager()` function. Changing this address, even with good intentions in mind, poses the problem of locking out of their shares the LPs in the process of redeeming. This happens since the storage mappings: `exitCycleId` and `lockedShares`, as well as the ERC-20 tokens (shares) of the LPs waiting for their withdrawal window will not have migrated to the new address, and will now be locked in the previous `withdrawalManager` proxy contract.

Recommendation

Remove the `setWithdrawalManager()` function or pause it by default.

Status

Addressed in the following PR:

https://github.com/maple-labs/pool-v2-private/pull/272

3S-MAPLE-M09

Malicious refinancer danger due to fixed term loan upgrades.

Id	3S-MAPLE-M09
Classification	Medium
Severity	High
Difficulty	High
Category	Suggestion
Relevant Links	fixed-term-loan-private/MapleLoan#L210

Description

In the previous `maple-core-v2` code, any address could be chosen for the refinancer when proposing new terms. The new versions of the fixed term loans add a check regarding the refinancer address, but place it in the `proposeNewTerms(...)` function, such that proposed terms in the previous loans version did not validate the refinancer address. The new check could lead to a false sense of security and delegates could accept new terms with a malicious refinancer address.

Exploit

- v400 fixed term loan borrower proposes refinancing with an invalid fee manager
- v400 loans are upgraded to v500 loans
- Given that v500 loans only allow whitelisted refinancers to be proposed, it could lead to a false sense of security
- The delegate accepts the new terms and the malicious refinancer address steals funds

Recommendation

The refinancer validation should be done on `acceptNewTerms(...)`. It would be possible to leave it on `proposeNewTerms(...)`, however, it would require the Maple team to track all the v400 maple loans (even the ones that were not funded / are idle) to check if there are invalid refinancer addresses before upgrading, and even then, attackers could still watch the *mempool* for the upgrade. Overall, it is safer to put the validation on `acceptNewTerms(...)`.

Status

Acknowledged by the team.

3S-MAPLE-L01

`MapleLoan`, `proposeNewTerms()` could have a check for duplicate selectors.

Id	3S-MAPLE-L01
Classification	Low
Category	Suggestion
Relevant Links	fixed-term-loan-private/MapleLoan#L210 open-term-loan-private/MapleLoan#L286

Description

Terms can be proposed with duplicated calls, which could lead to mistakes or phishing attacks.

Recommendation

Send the calls ordered by selector and check that the next selector is strictly bigger than the previous.

Status

Acknowledged by the team.

3S-MAPLE-L02

`MapleLoan`, `skim()` has no zero address transfer check.

Id	3S-MAPLE-L02
Classification	Low
Category	Bug
Relevant Links	open-term-loan-private/MapleLoan#L369 fixed-term-loan-private/MapleLoan#L454

Description

Could lead to loss of funds if someone sends a zero address by mistake.

Recommendation

Add a transfer to 0 address check.

Status

Addressed in the following PR:

https://github.com/maple-labs/open-term-loan-private/pull/49

3S-MAPLE-L03

In `MapleLoanInitializer`, the Borrower can choose a different `fundsAsset` than the lender.

Id	3S-MAPLE-L03
Classification	Low
Category	Bug
Relevant Links	open-term-loan-private/MapleLoanInitializer.sol#L56

Description

The code allows for the Borrower to choose a different `fundsAsset` than the lender. At the moment this will make `fund()` revert and the borrower has to redeploy the loan with the right argument, however this results in a lot of wasted gas and could even become a bigger problem in a future upgrade.

Recommendation

Add a check in the `MapleLoanInitializer` to make sure the loan's asset matches that of the loan manager's.

Status

Addressed in the following PR:

https://github.com/maple-labs/open-term-loan-private/pull/53

3S-MAPLE-L04

Borrowers can prevent `MapleLoanInitializer` from initializing a loan by using a vanity address.

Id	3S-MAPLE-L04
Classification	Low
Category	Bug
Relevant Links	open-term-loan-private/MapleLoanInitializer#L43-L54 fixed-term-loan-private/MapleLoanInitializer#L50-L65 pool-v2-private/PoolManagerInitializer#L43-L55

Description

By using the fallback function in the `MapleLoanInitializer` to initialize a loan (both for open term and for fixed term) it allows the borrower to choose a vanity address to end up in any public function of the `MapleLoanInitializer`, preventing the `_initialize()` to set the starting parameters.

This happens because the address of the borrower (which is not controlled by the protocol) is the first argument of the decode argument function which will mean that its first bytes will try to be matched to a function selector before ending up in the fallback function. The contract has several public functions (getters for the public variables in storage & others), so there are several combinations that could lead to this situation. This logic is also used throughout the repo in other initializer contracts.

At the moment, no one can take advantage of an uninitialized loan, due to all of the checks existing in the code, however, this is still a vulnerability of the deployment logic used throughout the repo to initialize contracts, and could lead to more significant exploits in future updates.

Recommendation

Have the first argument of the initializer be a protocol-controlled address, like the lender.

Status

Acknowledged by the team.

3S-MAPLE-L05

Consider using a time lock on critical permissioned functions.

Id	3S-MAPLE-L05
Classification	Low
Category	Suggestion

Description

In critical functions that may affect the decision of users to stay invested in the protocol (for example functions that set certain pool terms, fees, upgrades, etc.), there are no timelocks implemented, which will give no time for users to take their funds from the protocol and thus can deter them from interacting with Maple.

Recommendation

Implement timelocks in the types of functions mentioned above.

Status

Acknowledged by the team.

3S-MAPLE-L06

In `MapleGlobals`, there is no check in the `dataHash` argument when unscheduling a call.

Id	3S-MAPLE-L06
Classification	Low
Category	Suggestion
Relevant Links	globals-v2-private/MapleGlobals#L372-L375

Description

In `MapleGlobals.sol`, in `unscheduleCall()`, the `CallUnscheduled` event is emitting `dataHash` (the hash of the `callData_`) which is never checked, and could be different than the one it is unscheduling.

Recommendation

- emit the `dataHash` in storage (in the struct) and don't take the `callData_` as an argument to the function `unscheduleCall()`

or

- verify that the `dataHash` is valid before emitting by comparing it to the one in storage

Status

Addressed in the following PR:

https://github.com/maple-labs/globals-v2-private/pull/65

3S-MAPLE-L07

Consider adding a check to ensure that fees, delegate cover and so on have been explicitly set.

Id	3S-MAPLE-L07
Classification	Low
Category	Suggestion

Description

Some fees or delegate cover information are set in the globals. When fetched, there is no check that the values for a specific pool or loan manager have been set, so the default 0 value is assumed. This could lead to unexpected results, even more so if a pool is deployed through means other than the Maple frontend (or it does not behave as expected).

Recommendation

Create getters that check if the values were explicitly set, `isSet[key]`.

Status

Acknowledged by the team.

3S-MAPLE-L08

In `PoolManager`, `setOpenToPublic()`, there is no way to unset `openToPublic`.

Id	3S-MAPLE-L08
Classification	Low
Category	Suggestion
Relevant Links	pool-v2-private/PoolManager#L199

Description

The `setOpenToPublic(...)` function only enables setting `openToPublic` to true, so if a delegate sets a permissioned pool to open by mistake, it's impossible to undo it.

Recommendation

Add `setOpenToPublic` as an argument.

Status

Acknowledged by the team.

3S-MAPLE-L09

Throughout code-base: missing address checks.

Id	3S-MAPLE-L09
Classification	Low
Category	Bug
Relevant Links	maple-proxy-factory/MapleProxyFactory#L78 globals-v2-private/MapleGlobals#L120 fixed-term-loan-private/MapleLoanFeeManager#L46

Description

Whenever addresses change in the code, it's important to be careful about the possibility of setting the wrong address. The following occurrences could be dangerous in this regard

- `MapleProxyFactory`, `setGlobals(...)`
- `MapleGlobals`, `setMapleTreasury(...)`
- `Fixed-term-loan-private`, the `constructor` of `MapleLoanFeeManager`

Recommendation

Two options were suggested:

- use the `setPending-accept` pattern
- don't change addresses, given that the contracts are upgradeable

Status

Acknowledged by the team.

3S-MAPLE-N01

`MapleLoan` variables read from storage more than once.

Id	3S-MAPLE-N01
Classification	None
Category	Optimization
Relevant Links	open-term-loan-private/MapleLoan#L77 open-term-loan-private/MapleLoan#L162

Description

In the `MapleLoan` contract, the following variables are loaded from storage multiple times:

- `acceptNewTerms()`: `lender` is loaded twice from storage.
- `acceptNewTerms()`: `borrower` is loaded from storage three times.
- `makePayment()`: `lender` is loaded from storage three times.

Recommendation

These variables should be loaded from storage just once and placed in temporary memory for later use (within the same function).

Status

Addressed in the following PR:

https://github.com/maple-labs/open-term-loan-private/pull/48

3S-MAPLE-N02

`MapleLoanStorage` storage slot optimization.

Id	3S-MAPLE-N02
Severity	None
Difficulty	N/A
Category	Optimization
Relevant Links	open-term-loan-private/MapleLoanStorage#33

Description

On the `MapleLoanStorage` contract, a storage slot can be reduced by packing the last variable: `platformServiceFeeRate` (a uint64) with one of the addresses.

Recommendation

Pack the variables into a single storage slot.

Note: This may slightly harm code readability

Status

Acknowledged by the team.

3S-MAPLE-N03

`MapleLoan` change the order of `require` to save on gas.

Id	3S-MAPLE-N03
Classification	None
Category	Optimization
Relevant Links	open-term-loan-private/MapleLoan#L84 open-term-loan-private/MapleLoan#L88-L89

Description

On `open-term-loan/MapleLoan.sol`, line 84 of `acceptNewTerms()`, there is a `require` to check that the refinance commitment according to the terms sent by the user is valid and to do so `refinanceCommitment` is loaded from storage. Then, on lines 88 and 89 additional checks are made regarding the arguments previously used.

Recommendation

If these checks were to be done before, in the case of invalid arguments, it would save gas by not having to load `refinanceCommitment` from storage.

Status

Addressed in the following PR:

https://github.com/maple-labs/open-term-loan-private/pull/56

3S-MAPLE-N04

In `PoolManager`, `requestFunds(...)` repeated variable fetching from storage.

Id	3S-MAPLE-N04
Classification	None
Category	Optimization
Relevant Links	pool-v2-private/PoolManager#L221 pool-v2-private/PoolManager#L225

Description

On line 221 the `factory_` address is assigned from `IMapleProxied(msg.sender).factory()`. However, on line 225, `IMapleProxied(msg.sender).factory()` is called again instead of using the already fetched `factory_`.

Recommendation

Use the already fetched `factory_` variable.

Status

Addressed in the following PR:

https://github.com/maple-labs/pool-v2-private/pull/269

3S-MAPLE-N05

In `MapleLoanFactory`, `isLoan` has the same functionality of `isInstance` and thus can be removed.

Id	3S-MAPLE-N05
Classification	None
Category	Optimization
Relevant Links	open-term-loan-private/MapleLoanFactory#L12 open-term-loan-private/MapleLoanFactory#L24 maple-proxy-factory/MapleProxyFactory#L95

Description

The mapping `isLoan` is only modified in the respective `MapleLoanFactory.sol` contract which extends `MapleProxyFactory.sol`.
`isLoan` is only modified in line 24 of `MapleLoanFactory` where, in the same call, `isInstance` is also set to `true` in line 95 of `MapleProxyFactory`.
So every valid loan according to `isLoan` is a valid instance according to `isInstance` and every address that is not registered in `isLoan` will also not be registered in `isInstance`.

Recommendation

The mapping `isLoan` can be removed and `isInstance` can be used in its place whenever needed.

Status

Acknowledged, this was done to maintain compatibility with the fixed term loan factory (which is out of scope for our audit).

3S-MAPLE-N06

`LoanManager` code optimizations: no need to load payment struct from storage if loan is unimpaired.

Id	3S-MAPLE-N06
Classification	None
Category	Optimization
Relevant Links	[open-term-loan-manager-private/LoanManager#L361]()

Description

In the LoanManager contract, two changes can be made to reduce gas usage and improve readability:

In function `_accountForLoanImpairmentRemoval()`: loading the payment struct from storage should only be performed after the check to see if the loan is impaired, since the vast majority of the time, a payment will be made to an unimpaired loan, so there is no need to load this struct from storage as it will not be used. This change would result in a `makePayment()` call cheaper by around 3000 gas.

An identical issue can be found in function `_accountForLoanImpairment()`.

Recommendation

Only load the payment struct from storage after the line which `returns` if the loan is impaired.

Status

Addressed in the following PR:

```
https://github.com/maple-labs/open-term-loan-manager-private/pull/52
```

3S-MAPLE-N07

`LoanManager` code optimizations: redundant impairment functions.

Id	3S-MAPLE-N07
Classification	None
Category	Optimization
Relevant Links	open-term-loan-manager-private/LoanManager#L213-L225 open-term-loan-manager-private/LoanManager#L352-L358

Description

The LoanManager contract has the following code:

```
function impairLoan(address loan_) {
    bool isGovernor_ = msg.sender == _governor();
    (...)
    if (isGovernor_) {
        _accountForLoanImpairmentAsGovernor(loan_);
    } else {
        _accountForLoanImpairment(loan_);
    }
}

function _accountForLoanImpairment(address loan_) {
    impairedDate_ = _accountForLoanImpairment(loan_, false);
}

function _accountForLoanImpairmentAsGovernor(address loan_) {
    impairedDate_ = _accountForLoanImpairment(loan_, true);
}
```

In this code, functions `_accountForLoanImpairment(loan)` and `_accountForLoanImpairmentAsGovernor(loan)` add unneeded redundancy and make the code more cluttered, therefore more difficult to read and less gas efficient.

Recommendation

Function `impairLoan()` can call directly `accountForLoanImpairment(loan, isGovernor)`, and functions `_accountForLoanImpairment(loan)` and `_accountForLoanImpairmentAsGovernor(loan)` can be removed. Changing the same code to just:

```
function impairLoan(address loan_){
    bool isGovernor_ = msg.sender == _governor();
    (...)
    _accountForLoanImpairment(loan_, isGovernor_);
}
```

Note: This suggestion would require a minor change to `triggerDefault()`, sending false as the second function parameter on the call to function `_accountForLoanImpairment`.

Status

Acknowledged by the team.

3S-MAPLE-N08

Throughout code base: use Solidity native errors implementation instead of string errors.

Id	3S-MAPLE-N08
Classification	None
Category	Optimization

Description

Solidity allows for their implementation of native custom errors that is more gas-efficient than using a string within the `require` function to explain to the user where the call reverted (the way it's implemented in the Maple Protocol right now).

We understand that string errors might be used to give more information on where the call reverted; however, this information can still be passed to the user by having specifically named errors.

Recommendation

Use custom errors, for example, use `MapleLoan_AcceptBorrower_NotPendingBorrowerError()` instead of throwing `"ML:AB:NOT_PENDING_BORROWER"`.

Status

Acknowledged, would break Maple's convention so sticking with requires.

3S-MAPLE-N09

open-term-loan-private: check dateFunded!=0 first to save gas.

Id	3S-MAPLE-N09
Classification	None
Category	Optimization
Relevant Links	open-term-loan-private/MapleLoan#L162-L220 open-term-loan-private/MapleLoan#L173 open-term-loan-private/MapleLoan#L176 open-term-loan-private/MapleLoan#L397-L429

Description

On open-term-loan in function makePayment(), first the payment breakdown is called and after that the loan is checked to be active. getPaymentBreakdown() does several storage reads. dateFunded is not changed within the function scope, so the check if the loan is active (and the check for the principal as well) can be done before getPaymentBreakdown is called and save on gas from the storage reads in case the loan is inactive.

Recommendation

Change lines 173-178 to:

```
require(dateFunded != 0, "ML:MP:LOAN_INACTIVE");
require(principalToReturn_ <= principal, "ML:MP:RETURNING_TOO_MUCH");

(calledPrincipal_, interest_, lateInterest_, delegateServiceFee_,
platformServiceFee_) = getPaymentBreakdown(block.timestamp);
```

```
require(principalToReturn_ >= calledPrincipal_,
"ML:MP:INSUFFICIENT_FOR_CALL");
```

Status

Addressed in the following PR:

https://github.com/maple-labs/open-term-loan-private/pull/54

3S-MAPLE-N10

`pool-v2-private`: cache list length in memory and uncheck `i_` to save gas.

Id	3S-MAPLE-N10
Classification	None
Category	Optimization
Relevant Links	pool-v2-private/PoolManager#L179

Description

In the PoolManager contract, function `setIsLoanManager()`: variable `loanManagerList.length` could be held in memory preventing multiple storage reads during the `for` loop. The `++i` could also be unchecked for additional gas savings.

Recommendation

Cache the array length and uncheck the `++i.` This function could also receive a hint with the index of the loan manager in the `loanManagerList`, preventing the `for` loop that iterates through the list, resulting in additional gas savings.

Status

Acknowledged by the team.

3S-MAPLE-N11

Throughout code-base: Homogenize how access control is done.

Id	3S-MAPLE-N11
Classification	None
Category	Suggestion
Relevant Links	open-term-loan-manager-private/LoanManager#L39-L43 open-term-loan-private/MapleLoan#L38-L41 fixed-term-loan-manager-private/LoanManager#L996-L998

Description

There are some discrepancies in the way that checks for access control/requires are done throughout the code base. Sometimes a modifier is used, sometimes an internal function is used, and sometimes the `require` is just hard-coded. Some examples below:

- on `open-term/LoanManager.sol`
  - `isLoan` and `notPaused` are implemented as modifiers
  - `require` is used to check if factory or if PD (Pool Delegate) or PM (Pool Manager)
- on `fixed-term/LoanManager.sol`
  - internal function (`_requireCallerIsPoolDelegate`) is used to check if PD (different from open-term above)
  - internal function (`_requireProtocolNotPaused`) is used to check if protocol is not paused (different from open-term above)
- on `open-term/MapleLoan.sol` and on `fixed-term/MapleLoan.sol`
  - `whenNotPaused` is used as a modifier
  - `require` is used to check if factory, if borrower, if lender

We understand that requires might be used to return custom string error messages that specify, in addition to the contract, in which function it reverted (which nevertheless is not done in fixed-term `LoanManager` since internal functions are used here, where only the contract it reverts is specified in the error message); however, there is still a discrepancy on how the access control is done on the open-term and fixed-term for the `LoanManager` functions (internal functions vs requires).

Recommendation

We suggest using the same approach in both contracts. Also, there are three different implementations to check if a protocol is not paused (modifier `notPaused`, modifier `whenNotPaused`, internal function `_requireProtocolNotPaused`), we also suggest using the same approach across all contracts.

Additionally when a modifier is added to a function, the entire code of the modifier gets copied in all the functions. It is more gas efficient (on deployment) to implement the logic of the modifier in another function and call it from inside the modifier.

Status

Addressed in the following PRs:

maple-labs/fixed-term-loan-private#283

maple-labs/pool-v2-private#271

maple-labs/globals-v2-private#63

maple-labs/open-term-loan-private#55

maple-labs/open-term-loan-manager-private#53

maple-labs/fixed-term-loan-manager-private#31

3S-MAPLE-N12

Missing documentation for `MapleLoan`, `SetPendingLender` and `AcceptLender` being implemented on `MapleLoan` but not on `LoanManager`.

Id	3S-MAPLE-N12
Classification	None
Category	Suggestion

Description

Fixed and Open term loans implement `SetPendingLender` and `AcceptLender`; however, the corresponding LoanManagers don't. This is a design choice to allow a future upgrade of `LoanManager` without the necessity of upgrading `MapleLoan`.

Status

Addressed in the following PR:

https://github.com/maple-labs/open-term-loan-manager-private/pull/56

3S-MAPLE-N13

Multiple documentation fixes throughout the repository.

Id	3S-MAPLE-N13
Classification	None
Category	Suggestion

Description

Throughout the repository, a total of 14 comments and 21 wiki errors were reported.

Recommendation

Fix the corresponding errors.

Status

Addressed in the following PRs:

https://github.com/maple-labs/open-term-loan-private/pull/50
https://github.com/maple-labs/open-term-loan-manager-private/pull/50
https://github.com/maple-labs/open-term-loan-manager-private/pull/55

3S-MAPLE-N14

`globals-v2-private`: typo in function name.

Id	3S-MAPLE-N14
Classification	None
Category	Suggestion
Relevant Links	globals-v2-private/MapleGlobals#L174

Description

There is a typo in the name of the function `setContactPause()`.

Recommendation

Change `setContactPause()` to `setContractPause()`.

Status

Addressed in the following PR:

maple-labs/globals-v2-private#61