CANTINA

Maple Finance

Security Review

A Cantina Managed review by:

Christoph Michel, Lead Security Researcher

Riley Holterhus, Lead Security Researcher

Jonatas Martins, Associate Security Researcher

June 5, 2023

Contents

1 Introduction

1.1 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.2 Risk assessment

Severity	Description
Critical	*Directly* exploitable security vulnerabilities that need to be fixed.
High	Security vulnerabilities that may not be directly exploitable or may
	require certain conditions in order to be exploited.
	All high issues should be addressed.
Medium	Objective in nature but are not security vulnerabilities.
	Should be addressed unless there is a clear reason not to.
Low	Subjective in nature.
	They are typically suggestions around best practices or readability.
	Code maintainers should use their own judgment as to whether to address such issues.
Gas	Suggestions around gas saving practices
Informational	Suggestions around best practices or readability.

1.2.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. When determining the severity one first needs to determine whether the finding is subjective or objective. All subjective findings are considered of Minor severity.

Next it is determined whether the finding can be regarded as a security vulnerability. Some findings might be objective improvements that need to be fixed, but do not impact the project's security overall (Medium).

Finally, objective findings of security vulnerabilities are classified as either critical or major. Critical findings should be directly vulnerable and have a high likelihood of being exploited. Major findings on the other hand may require specific conditions that need to be met before the vulnerability becomes exploitable.

2 Security Review Summary

Maple Finance is an institutional crypto-capital network built on Ethereum and Solana. 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.

From April 24th to May 5th the Cantina team conducted a review of maple-core-v2-private on commit hash d3409c29.

The reviewed code is a new release of Maple V2 that implements Open Term Loans. It also includes enhancements to global contracts to allow for greater flexibility in the allowlist, improvements to incident response, and protection for deployments. Finally, it includes adjustments to Fixed Term Loan contracts to work with the new architecture.

Submodule	**Commit hash**
fixed-term-loan-private	v5.0.0-rc.1
fixed-term-loan-manager-private	v3.0.0-rc.1
globals-v2-private	v1.1.0-rc.1
liquidations-private	v2.0.0
open-term-loan-private	v1.0.0-rc.1
open-term-loan-manager-private	v1.0.0-rc.1
pool-v2-private	v2.0.0-rc.1
withdrawal-manager-private	v1.0.0

The team identified a total of **18** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 1
- Medium Risk: 1
- Low Risk: 6
- Gas Optimizations: 1
- Informational: 9

3 Findings

3.1 High Risk

3.1.1 Borrower can choose Loan migration arguments or perform a noop migration

Severity: High Risk

Context: `fixed-term-loan/MapleLoan`, `open-term-loan/MapleLoan`

Description: The `MapleLoan.upgrade(uint256 toVersion_, bytes calldata arguments_)` function can be called by the borrower (and the security admin). The borrower can choose migration arguments for future migrations that are in their favor, like increasing principal or decreasing interest rates. The borrower can also skip running the migration code but still upgrade to the latest loan version by encoding arguments that would not call the `fallback` function on the migrator. For example, for the old MapleLoanV4Migrator they could have chosen arguments that call `encodeArguments(0)`, skipping the migration code in the fallback (the proxy would still point to the new implementation code).

Recommendation: There is incentive for the borrower to choose migration arguments in their favor or "deny" a migration by encoding a different function than the fallback. For better control, consider removing `arguments` as a parameter for `MapleLoans` and always call `IMapleProxyFactory(_factory()).upgradeInstance(toVersion_, "")` without arguments. The arguments should be hardcoded in the migrator contract. Alternatively, consider restricting `upgrade` access to only the security admin.

Maple: Fixed in PR-61 (open-term loan) and PR-291 (fixed-term loan) by restricting upgrades to only the security admin.

Cantina: Fixed.

3.2 Medium Risk

3.2.1 Reentrant tokens should not be allowed by governance

Severity: Medium Risk

Context: `open-term-loan`

Description: The protocol doesn't follow the checks-effects-interactions pattern and performs transfers in the middle of it. This can lead to borrowers or other untrusted third parties receiving a callback in the middle of the execution while the contract is in an inconsistent state. These callbacks can happen when using funds tokens that support an ERC777-style transfer callback. While the `LoanManager` has some reentrancy guards, this is not enough to fully protect against all reentrancy issues that can span across several contracts (`Pool`, `PoolManager`, `LoanManager`, `Loan`) that are involved in a single call.

Examples:

- The transfer in `OTLoan.makePayment` happens in the middle of the function, after state updates and before the `LoanManager.claim` call. Inside `LoanManager.claim` the `LoanManager` reads the loan's state again:

```
// this value might not be the one you expect from the call, might have been changed twice in `makePayment`
↪  already
uint256 principalRemaining_ = ILoanLike(msg.sender).principal();

// Calculate the original principal to correctly account for removing `unrealizedLosses` when removing the
↪  impairment.
uint256 originalPrincipal_ = uint256(_int256(principalRemaining_) + principal_);

_accountForLoanImpairmentRemoval(msg.sender, originalPrincipal_);
```

One can break the impairment accounting this way by:

1. `makePayment(principalToReturn_ = principal - 1)`. Get control at the transfer here
2. Reenter with `makePayment(principalToReturn_ = 0)`.
3. claim is called and recomputes `principalToReturn_ + L.principal() = 0 + 1 = 1`
4. reduces loan impairment by a much smaller amount, LM keeps unrealized losses

There might be more severe issues like the pool receiving the interest and principal payments from a transfer while `LoanManager`'s `prinicipalOut` is not decreased yet, resulting in an over-approximation of pool assets.

Recommendation: Tokens with callbacks should not be listed as funds or collateral assets by governance.

Maple: Business said that there are no plans for a token that implements a callback, so we'll take no action for now.

Cantina: Acknowledged.

3.3 Low Risk

3.3.1 `canDeploy` functions can revert instead of returning `false`

Severity: Low Risk

Context: `globals/MapleGlobals.sol#L479`

Description: If `_canDeployFrom[factory_][caller_]` is found to be false in the `canDeploy` and `canDeployFrom` functions, there will be an additional check if `factory_` is a loan manager factory and `caller_` is a pool manager. However, the check for if `caller_` is a pool manager assumes that `caller_` is a contract that implements a `factory()` function:

```
function _isPoolManager(address contract_) internal view returns (bool isPoolManager_) {
    address factory_ = IProxyLike(contract_).factory();

    isPoolManager_ = (isInstanceOf["POOL_MANAGER_FACTORY"][factory_]) &&
    ↪  IProxyFactoryLike(factory_).isInstance(contract_);
}
```

This is not the case for EOAs and other contracts, so this function will revert instead of returning false.

Recommendation: To make sure that `true` or `false` is always returned and reverts never happen, there are two recommended options:

1. Rework the `_isPoolManager` function to have more checks against the `factory()` function reverting or not returning an address:

```
function _isPoolManager(address caller_) internal view returns (bool isPoolManager_) {
    (bool success, bytes memory returndata) =
    ↪  caller_.staticcall(abi.encodeWithSelector(IProxyLike.factory.selector));
    if (success && returndata.length == 32) {
        uint256 decodedFactoryAsUInt_ = abi.decode(returndata, (uint256));
        if (decodedFactoryAsUInt_ <= type(uint160).max) {
            address factory_ = address(uint160(decodedFactoryAsUInt_));
            isPoolManager_ = (isInstanceOf["POOL_MANAGER_FACTORY"][factory_]) &&
            ↪  IProxyFactoryLike(factory_).isInstance(caller_);
        }
    }
}
```

2. Slightly rework the deployment process so the `_isPoolManager` helper function is not needed. One way to do this would be to allow pool managers to set `_canDeployFrom` of the loan managers they are about to deploy.

Maple: Acknowledged.

Cantina: Acknowledged. If future updates further use the `_isPoolManager` function, please keep this behavior in mind.

3.3.2 Fixed-term loan manager refinancing does not check borrower is valid

Severity: Low Risk

Context: `fixed-term-loan-manager/LoanManager.sol#L134`

Description: It is possible for a valid borrower to become invalid through the governor calling the `setValidBorrower` function. Previously, a fixed-term loan refinancing would check the validity of the current borrower in the `_validateAndFundLoan` function on the `PoolManager`. This check no longer takes place, which means an invalid borrower can receive funds when they previously wouldn't have been able to. This also does not match the behavior of open-term loan refinancing, which checks the validity of the borrower in the `claim` function when funds are being transferred.

Recommendation: Disallow sending funds to borrowers who have had their validity removed. This can be accomplished by adding these lines to the `acceptNewTerms` function in the fixed-term loan manager:

```
if (principalIncrease_ > 0) {
+   address borrower_ = ILoanLike(loan_).borrower();
+   require(IMapleGlobalsLike(_globals()).isBorrower(borrower_), "LM:ANT:INVALID_BORROWER");
    IPoolManagerLike(poolManager).requestFunds(loan_, principalIncrease_);
}
```

Maple: Fixed in PR-37.

Cantina: Fixed.

3.3.3 Accepting new terms while loan is not funded leads to wrong principal transfers

Severity: Low Risk

Context: `open-term-loan/MapleLoan.sol#L99`

Description: For open-term loans, the **requested** principal is stored in the `principal` storage variable upon initialization (instead of an extra `principalRequested` variable lke for fixed-term loans). This leads to issues when refinancing a loan that hasn't been funded yet. The `principal` variable is still treated as if the principal was already paid to the borrower in the case of an unfunded loan. When increasing the principal via the refinancing terms the borrower only receives the increase, not the original principal. If the principal is decreased, the borrower has to pay the difference in principal for the "loan".

Recommendation: Consider requiring `dateFunded != 0` for `acceptNewTerms` to exclude this edge case. If terms need to be renegotiated before the loan is even funded, the borrower can just create a new loan with the new terms.

Maple: Fixed in PR-62

Cantina: The Maple team has decided to add `dateFunded != 0` in `proposeNewTerms`. Since `acceptNewTerms` can't successfully execute before `proposeNewTerms` is called, this is equivalent to the proposed solution. Fixed.

3.3.4 Impairments and calls cleared through refinancing does not emit events

Severity: Low Risk

Context: `open-term-loan/MapleLoan.sol#L147`

Description: When the borrower accepts new terms for an open-term loan, any pending impairment or principal call is cleared. However, this implicit clearing of the states in `acceptNewTerms` does not emit the `ImpairmentRemoved` or `CallRemoved` events that would be emitted if the states were cleared via `removeImpairment` or `removeCall`.

Recommendation: Consider emitting these events or be aware that any `NewTermsAccepted` event also acts as an impairment and call clear event.

Maple: Whilst we opted not to add events in the end we have added a note in both fixed-term and open-term loans to make it more clear that impairments/ calls get cleared.

Cantina: Acknowledged.

3.3.5 Pending refinance commitments after clearing loan accounting

Severity: Low Risk

Context: `open-term-loan/MapleLoan.sol#L469`, `fixed-term-loan/MapleLoan.sol`

Description: The `MapleLoan._clearLoanAccounting` function does not clear the `refinanceCommitment`. For open-term loans, the borrower can accept new terms and trigger a refinance with the old commitment even if the loan was closed because all principal was paid back or because it defaulted. This fails in the `LoanManager.claim` function due to the `isLoan(msg.sender)` modifier not being valid anymore, negating the impact.

POC Shows the revert in `LoanManager.claim` and uses as base the CallPrincipal.t.sol. To run it copy the function to `CallPrincipalTests`.

```
function test_acceptNewTerms_afterMakeFullPayment() external {
    // Warp to exactly the payment due date
    vm.warp(start + paymentInterval / 2);

    uint256 principalDiff = 1e18;

    // Create a call to increase the principal
    bytes[] memory calls = _encodeCall(abi.encodeWithSignature("increasePrincipal(uint256)", principalDiff));
    uint256 deadline     = block.timestamp + 10 days;

    vm.prank(poolDelegate);
    loanManager.proposeNewTerms(address(loan), address(openTermRefinancer), deadline, calls);


    (
        ,
        uint256 interest_,
        uint256 lateInterest_,
        uint256 delegateServiceFee_,
        uint256 platformServiceFee_
    ) = loan.getPaymentBreakdown(block.timestamp);

    //Get the total of interests
    uint256 totalPayment = interest_ + lateInterest_ + delegateServiceFee_ + platformServiceFee_;

    // Mint the borrower the principal + partial payments to make the full payment
    fundsAsset.mint(borrower, totalPayment + principal);

    vm.startPrank(borrower);
    fundsAsset.approve(address(loan), totalPayment + principal);
    //@audit 1. Borrower pays all principal before accept new terms
    loan.makePayment(principal);

    assertEq(loan.paymentInterval(), 0);
    assertEq(loan.dateCalled(),      0);
    assertEq(loan.dateImpaired(),    0);
    assertEq(loan.datePaid(),        0);
    assertEq(loan.paymentDueDate(),  0);
    assertEq(loan.principal(),       0);

    //@audit 2. Fail here because the loan is removed from LM
    vm.expectRevert("LM:NOT_LOAN");
    loan.acceptNewTerms(address(openTermRefinancer), deadline, calls);

    vm.stopPrank();
}

function _encodeCall(bytes memory call) internal pure returns (bytes[] memory calls) {
    calls = new bytes[](1);
    calls[0] = call;
}
```

Recommendation: We still recommend properly resetting all non-essential loan storage variables like `refinanceCommitment`, `delegateServiceFeeRate` and `platformServiceFeeRate` in `_clearLoanAccounting` as the Loan contract itself should be consistent and not rely on the `LoanManager` to revert if accepting new terms after the loan has been paid off.

Maple: Fixed in PR-64.

Cantina: Fixed.

3.3.6 Interest rate decimal change can break external integrations

Severity: Low Risk

Context: `fixed-term-loan/MapleLoanV5Migrator.sol`

Description: The `MapleLoanV5Migrator` migrates all interest rates from 18 decimals to 6 decimals. The public getter functions for these also return the rates with the new decimals, breaking existing contract integrations or off-chain dashboards.

Recommendation: Ensure that this change is communicated to external integrators.

Maple: We talked with business and they confirmed that no integrator is relying on querying the contract state directly, as we offer an API/SDK.

Cantina: Acknowledged.

3.4 Gas Optimization

3.4.1 For-loop optimization

Severity: Gas Optimization

Context: pool/PoolManager, pool/PoolDeployer

Description: The for-loop in PoolManager#setIsLoanManager, PoolDeployer#deployPool, and PoolDeployer#getDeploymentAddresses are the only ones not optimized in the codebase.

Recommendation: We recommend optimizing the for-loops

Maple: Fixed in PR-292, PR-68, PR-63

Cantina: Maple team decided to remove for-loop optimization for better readability in loops that don't necessarily need to be optimized. Fixed.

3.5 Informational

3.5.1 Open-term loan defaults can be simplified

Severity: Informational

Context: open-term-loan-manager/LoanManager.sol#L264

Description: Since open-term loans do not have collateral, the repossess/liquidation logic will only be relevant if someone sends tokens manually into the loan contract. This should never occur, and even if it does, the governor or borrower would still be able to withdraw these tokens at any point through the skim function.

Recommendation: Consider simplifying the liquidation/repossession logic for open-term loans. By assuming that the loan holding tokens will almost never happen and can be dealt with externally if it does, the _distributeLiquidationFunds function can be entirely removed, and the repossess function can be simplified.

Maple: Acknowledged. It is possible that future loan types will also use the open-term loan manager, so it makes sense to keep it generic.

Cantina: Acknowledged.

3.5.2 isFactory incorrect comment

Severity: Informational

Context: globals/MapleGlobals.sol#L428

Description: The isFactory function has a comment that claims the liquidator factory checks isFactory("LOAN_MANAGER", address(this));. However, the liquidator factory actually calls isFactory("LOAN_MANAGER", IMapleProxied(msg.sender).factory()).

Recommendation: Update the comment to reflect that isFactory is not checking that address(this) is the loan manager factory.

Maple: Fixed in PR-67.

Cantina: Fixed.

3.5.3 Open-term loan manager functions missing isLoan validation

Severity: Informational

Context: open-term-loan-manager/LoanManager.sol#L33

Description: The following open-term loan manager functions do not validate their loan_ argument using the isLoan modifier:

- proposeNewTerms
- rejectNewTerms
- callPrincipal
- removeCall

This allows the pool delegate to forward a call to an arbitrary address.

Recommendation: For extra safety, add the `isLoan` modifier to the four functions mentioned.

Maple: Fixed in PR-59.

Cantina: Fixed.

3.5.4 Inconsistent `PRECISION` between loan managers

Severity: Informational

Context: `fixed-term-loan-manager/LoanManager`, `open-term-loan-manager/LoanManager.sol#L27`

Description: The `PRECISION` constant variable is declared with different values in `fixed-term-loan-manager/LoanManager` and `open-term-loan-manager/LoanManager` contracts, which could lead to issues in integrating contracts

Recommendation: We recommend using the same value for both contract

Maple: Thanks for pointing it out. We ended up deciding not to consolidate on a single precision a few weeks back.

Cantina: Acknowledged.

3.5.5 Consistent naming for `lateInterestPremiumRate_`

Severity: Informational

Context: `fixed-term-loan/MapleLoan.sol#L755`, `fixed-term-loan/MapleLoan.sol#L782`, `fixed-term-loan/MapleLoan.sol#L844`

Description: The `lateInterestPremium_` was renamed to `lateInterestPremiumRate_` but not in `_getLateInterest`, `_getPaymentBreakdown` and `_getRefinanceInterest`.

Recommendation: Consider changing the name to `lateInterestPremiumRate_` everywhere.

Maple: Fixed in PR-293

Cantina: Fixed.

3.5.6 Pool's pause control is not on a per-function level

Severity: Informational

Context: `pool/PoolManager`

Description: All contracts can be paused and individual functions can be unpaused through governance, except for the Pool contract. The pool contract forwards the decision to the `PoolManager`'s `canCall` function which performs a single pause check for all pool functions:

```
function canCall(bytes32 functionId_, address, bytes memory data_)
    external view override returns (bool canCall_, string memory errorMessage_)
{
    if (IGlobalsLike(globals()).isFunctionPaused(msg.sig)) return (false, "PM:CC:PAUSED");
    // ...
}
```

Recommendation: If fine-grained pause control over the pool's functions is desired, consider code like this:

```
if (
    functionId_ == "P:redeem"          ||
    functionId_ == "P:withdraw"        ||
    functionId_ == "P:removeShares"    ||
    functionId_ == "P:requestRedeem"   ||
    functionId_ == "P:requestWithdraw"
) {
    if (IGlobalsLike(globals()).isFunctionPaused(msg.sender, _convertPoolFunctionIdToSelector(functionId_))) {
        return (false, "PM:CC:FN_PAUSED");
    }
    return (true, "");
}
```

where `_convertPoolFunctionIdToSelector(bytes32) -> bytes4` converts the 32-byte function ID to its actual 4-byte solidity function selector.

Maple: We are aware of this issue, but we opted to not address it for this release.

Cantina: Acknowledged.

3.5.7 `PoolManager._getLoanManager(loan)` does not check if loan is valid

Severity: Informational

Context: `pool/PoolManager`

Description: While the `_getLoanManager(loan)` function checks that `loan.lender()` is a valid `LoanManager`, it does not verify that `loan` itself is a valid loan. A fake loan contract could return valid loan managers. The impact is negligible because all functions currently using `_getLoanManager(loan)` will call a function on the `LoanManager` with the `loan` specified as a parameter that would revert for invalid loans.

Recommendation: Consider checking that `loan` itself is a valid loan by verifying that its `loan.factory()` is a valid OT or FT factory and then checking `factory.isLoan(loan)`.

Maple: This issue also came up in one of our internal audits and we decided not to act upon. Like you mentioned, the impact is negligible, so we prefer not to change the code at this point.

Cantina: Acknowledged.

3.5.8 Ambiguous negation function naming

Severity: Informational

Context: `pool/PoolManager`, `pool/PoolManager`

Description: For some `revert` and modifier function names it's unclear to what clauses the "Not" negation refers to:

- `onlyIfNotConfiguredOrPoolDelegate`: Here, the `Not` refers only to the first clause (`if !configured || poolDelegate`)
- `_revertIfNotPoolDelegateOrGovernor`: Here, the `Not` refers to both clauses (`if !(poolDelegate || governor)`)

Recommendation: Consider using less ambiguous names like `onlyIfPoolDelegateOrNotConfigured` or `_revertIfNeitherPoolDelegateNorGovernor`.

Maple: Fixed in PR-275 and PR-276.

Cantina: Fixed.

3.5.9 Open-term loan differences with documentation

Severity: Informational

Context: Open-Term-Loan Docs

Description: There are some differences between the open-term loan documentation and its code:

> If PD doesn't post minimum required Cover, service and management fees are routed to the Pool. - Source

They are routed to the treasury/pool.

> If the loan is called by the Pool Delegate(PD), the borrower must repay the requested amount. After PD calls a loan, the borrower can either repay in full within Notice Period or allow PD to default the loan. - Source

With the exception of the borrower accepting new loan terms, in which case the loan call is cleared and only the old interest & fees and any potential loan decrease is paid back.

> Governor/ PD can revert a loan impairment anytime. - Source

PD cannot revert loan impairments initiated by the governor.

> The PoolDelegate directly `callsPrincipal()` with the principal amount, this then calls the Loan's `callPrincipal()`. No state changes occur on the LoanManager. - Source

The PD calls `callPrincipal()` instead of `callsPrincipal()` on the LM.

Recommendation: We believe the differences in the docs are mostly due to ambiguous wording and to keep descriptions short. Check if the differences are indeed intended and consider resolving these.

Maple: We've updated to docs with the recommendations, we'll keep this in mind as we write up the public facing docs as well.

Cantina: Fixed.