macro

Maple Finance A-2

Security Audit

August 14, 2024
Version 1.0.0

Presented by 0xMacro

Table of Contents

Introduction

This document includes the results of the security audit for Maple Finance's smart contract code as found in the section titled ‘Source Code’. The security audit was performed by the Macro security team from August 5, 2024 to August 8, 2024.

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

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

Overall Assessment

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

Severity	Count	Acknowledged	Won't Do	Addressed
Low	1	1	-	-
Code Quality	2	-	-	2

Maple Finance was quick to respond to these issues.

Specification

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

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

Source Code

The following source code was reviewed during the audit:

Initial:

- syrup-utils-private @ b3d1bc1baebd4597181c626101a8b66dcddadbf7
- mpl-migration @ 3d49ab895cf4a13d26d3331a004aacb7858d9877
- open-term-loan-private @ f71cc276f88a98cccd7b72ce8333b0f7477f2c87
- fixed-term-loan-private @ 622c3b5d14799bfca02f9f59dd743fc13c4a9cb4

Final:

- syrup-utils-private @ 1a67384918d63e2f763a3cb087f991acc29918b4

Specifically, we audited the following contracts within **syrup-utils-private** repository:

Contract	SHA256
contracts/MplUserActions.sol	`7d9b3dc11f4468062e2b23deae9841ffc632a5370bd8ee8520142df61dd89f54`
contracts/SyrupDrip.sol	`8e0431604376ba3133c2e05fcc6f69d3098c328f769ffe0a96f41fe0dea1e413`
contracts/SyrupUserActions.sol	`7f9b3c66c3313d607a0e7500e3edc8cd3bfb23868217b83779d369ad2d9b1f80`

We also audited the following contract within **mpl-migration** repository:

Contract	SHA256
contracts/Migrator.sol	`98be3049ca0fbfab8ed40349149700b6a1f26fb851d2286dde1415ea19894227`

We audited the following contracts within **open-term-loan-private** repository:

Contract	SHA256
contracts/MapleLoan.sol	`5ef6f91039b48e5a4954604475aa543f9cc61913fe4c004a66307c54be86bd3c`
contracts/MapleLoanStorage.sol	`5b180d68121a44cea01abcecdbee47af3940f98cf95b52d06c5fcba9c22618d9`

We audited the following contracts within **fixed-term-loan-private** repository:

Contract	SHA256
contracts/MapleLoan.sol	`917555370d304ccb744ec1b49077a988b6fb409c73574ea15ae313e68d544db1`
contracts/MapleLoanStorage.sol	`54756f20d7fcbfa18eab5518f246acd04d754dc8a0a998064f053e36bff2ddfd`

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

Issue Descriptions and Recommendations

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

Security Level Reference

We quantify issues in three parts:

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

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

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

Issue Details

L-1 Conditional use of user permit signature

TOPIC	STATUS	IMPACT	LIKELIHOOD
Edge Case	Acknowledged	Medium	Low

Contracts `MplUserActions` and `SyrupUserActions` implement an internal `_permit()` function in all `xWithPermit()` functions on both contracts to handle the external `permit()` call only if the current allowance is less than the amount required:

```
function _permit(
  address asset_,
  uint256 deadline_,
  uint256 amount_,
  uint8 v_,
  bytes32 r_,
  bytes32 s_
) internal {
  uint256 allowance_ = IERC20Like(asset_).allowance(
    msg.sender,
    address(this)
  );

  if (allowance_ < amount_) {
    IERC20Like(asset_).permit(
      msg.sender,
      address(this),
      amount_,
      deadline_,
      v_,
      r_,
      s_
    );
```

```
  }
}
```

Reference: MplUserActions.sol#L127-133

If the allowance covers the amount intended to be transferred, users' valid signatures will not be used, nor will their nonce be set, potentially allowing the same function to be executed twice with the same parameters. Although not likely, if the front end or client mistakenly sends the transaction twice with the same data, this call could succeed with the same passed signature.

Allowance checks could be abstracted from the smart contracts layer, avoiding unnecessary logic execution and signature generation if not required.

Remediations to Consider:

Consider using `try-catch` with the `permit()` call to prioritize using the user's signature and nonce and checking the allowance if the request reverts.

RESPONSE BY MAPLE FINANCE

> Acknowledged - We will ensure that the frontend checks the allowance and doesn't accidentally sent two transactions as the implementation optimises for a denial of service.

~~Q-1~~ `asset`, `globals`, and `stakedSyrup` variables can be immutable

TOPIC	STATUS	QUALITY IMPACT
Code quality	Fixed	Low

In `SyrupDrip` contract, the `asset`, `global`, and `stakedSyrup` variables are only set in the contract's `constructor()` function. Consider declaring these variables as `immutable`.

~~Q-2~~ No direct retrieval method for `id` status

TOPIC	STATUS	QUALITY IMPACT
Use Cases	Fixed	Low

`SyrupDrip` contract sets each specific claim ID in the `bitmap` mapping. However, there are no direct methods to fetch a specific ID's status. Consider adding an external view function to allow users and integrators to verify the status of specific IDs in the `bitmap` mapping.

Disclaimer

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

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

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