macro

Maple Finance A-6

Security Audit

September 18, 2025
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 September 15, 2025 to September 18, 2025.

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

Maple Finance was quick to respond to these issues.

Specification

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

- Discussions on Telegram with the Maple Finance team.

Trust Model, Assumptions, and Accepted Risks (TMAAR)

The `GovernorTimelock` contract is a role-based governance system that enforces delays on action execution, protected by separating responsibilities among multiple actors who each fulfill different key functions in the governance process.

- Minimum delay (`MIN_DELAY`) is set to 1 day, after which proposals becomes executable.
- Minimum execution window (`MIN_EXECUTION_WINDOW`) is set to 1 day, after which proposals becomes invalid.
- Default delay and execution window can be configured (`setDefaultTimelockParameters()`), but must respect the enforced minimum values.
- Custom target/selector delays can be set (`setFunctionTimelockParameters()`), but must respect the enforced minimum values.

- When custom function timelock parameters are proposed(`setFunctionTimelockParameters()`), the previously set values are used as delays.
- Both default and custom delay and execution window settings must go through the full proposal cycle (scheduling by a proposer, waiting for the required delay period, and execution by an Executor).

Actors

Proposer (`PROPOSER_ROLE`)

The Proposer, intended to be the Governor Multisig, is a trusted role responsible for scheduling governance proposals. This actor is trusted to submit legitimate proposals, but the contract is designed to limit its power.

- Can schedule new governance proposals, which are subject to a mandatory time delay.
- Cannot execute, cancel, or directly schedule proposals that modify roles.
- If compromised, the primary mitigation is the Canceller role, which can unschedule malicious proposals, and the Role Admin, which can revoke the Proposer's role.

Executor (`EXECUTOR_ROLE`)

The Executor, intended to be an Executor Admin and the Operational Admin, is trusted to enact valid proposals. This role is the final step in the governance process, executing decisions after their time delay has passed and before the execution window.

- Can execute proposals that have passed their time delay and have not been cancelled.
- Cannot schedule or cancel proposals. The contract verifies proposal data at execution time to prevent tampering.

- A compromised Executor cannot create malicious proposals. Their ability to do harm is limited to executing malicious proposals scheduled by a compromised Proposer in the worst case. The primary mitigation is the Role Admin, which can revoke the Executor's role.

Canceller (`CANCELLER_ROLE`)

The Canceller, intended to be a Security Admin, is a highly trusted safety role. It serves as a critical check on the Proposer, responsible for preventing malicious proposals from being executed.

- Can unschedule active proposals to prevent their execution.
- By design, cannot cancel role update proposals. This is a key recovery mechanism, preventing a rogue Canceller from blocking the removal of a compromised actor.
- If compromised, the canceller can disrupt governance by cancelling legitimate proposals, halting progress until their role is revoked by the Role Admin.

Role Admin (`ROLE_ADMIN`)

The Role Admin, intended for the Governor Multisig and a dedicated Role Admin, holds administrative control over the role management. This is a highly trusted role responsible for managing the security and composition of the other roles, but limited to proposing and using the regular proposal cycle.

- Can propose to grant or revoke any role, including its own.
- Cannot execute its own proposals. Role updates are subject to the standard time delay and must be executed by an Executor as any othe proposal.
- If compromised, Executors serve as the final line of defense, as they are trusted not to execute malicious role change proposals. Another Role Admin can then revoke the compromised admin's access, relying on the integrity of non-malicious Executors to approve only legitimate role changes.

Token Withdrawer

The Token Withdrawer is a trusted role with the singular ability to withdraw any ERC20 tokens held by the `GovernorTimelock` contract. This role is intended for treasury management or recovering tokens sent to the contract.

- Can withdraw any ERC20 token to its own address at any time.
- The role is not managed by the `ROLE_ADMIN`. It can only be changed via a two-step process: the current withdrawer proposes a new one, and the new address must call a function to accept the role.

Source Code

The following source code was reviewed during the audit:

- **Repository:** globals-v2
- **Commit Hash (initial):** `c19ed92f23af4903a0d2781f523394ea57936c13`
- **Commit Hash (final):** `42136fbec446d397ce553dd72c48e8e06f88930f`

Specifically, we audited the following contracts within this repository:

Source Code	SHA256
contracts/GovernorTimelock.sol	`618801abdc091928b8705df4c3371ff7ef49087822931f53cf5185381d853322`
contracts/interfaces/IGovernorTimelock.sol	`54583995a6c2967a7afa492926c81095e80662634fc988f4eedfe2bed43000b6`

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~~ Compromised tokenWithdrawer cannot be recovered

TOPIC	STATUS	IMPACT	LIKELIHOOD
Protocol Design	Fixed	Medium	Low

In the `GovernorTimelock`, only the current `tokenWithdrawer` can propose a new pending `tokenWithdrawer`.

```
function setPendingTokenWithdrawer(address newPendingTokenWithdrawer_) exter
    require(msg.sender == tokenWithdrawer, "GT:SPTW:NOT_AUTHORIZED");

    pendingTokenWithdrawer = newPendingTokenWithdrawer_;

    emit PendingTokenWithdrawerSet(newPendingTokenWithdrawer_);
}
```

Therefore, if the current `tokenWithdrawer` is compromised, it would not be possible to replace it without redeploying the contract.

Remediations to consider

- Update `setPendingTokenWithdrawer()` to use the `GovernorTimelock` proposal mechanism or use the existing and available role-based access control to grant permission access to the `setPendingTokenWithdrawer()`.

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.