Term Finance

Term Finance – Smart Contract Changes (Part 2)

Security Assessment Report

Version: 2.0

November, 2023

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of a selected list of changes made to the Term Finance smart contracts.

The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the Term Finance smart contracts contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Term Finance smart contracts.

Overview

Term Finance is a noncustodial fixed-rate liquidity protocol modeled on tri-party repo arrangements common in traditional finance.

Liquidity suppliers and takers are matched through a unique weekly auction process where liquidity takers submit bids and suppliers submit offers to the protocol, which then determines an interest rate that clears the market.

Bidders who bid more than the clearing rate receive liquidity and lenders asking less than the clearing rate, supply.

Security Assessment Summary

This review was conducted on the files hosted on the term-finance repository, with majority of the code changes assessed at the commit tagged `0.5.6` (3bac4c5).

The scope of this assessment was strictly limited to the code changes related to the following PRs:

- `PR 931`
- `PR 947`
- `PR 954`
- `PR 981`
- `PR 948`
- `PR 1004`
- `PR 1045`
- `PR 1056`
- `PR 1072`
- `PR 1074`
- `PR 1097`
- `PR 1098`
- `PR 956`
- `PR 977`
- `PR 980`
- `PR 988`
- `PR 990`
- `PR 1018`
- `PR 1066`
- `PR 1121`

Note, `PR 1097`, `PR 1098` and `PR 1121` were assessed at commit f214c85 of term-finance-contracts repository, which aside from changes strictly related to these PRs, is identical to commit 3bac4c5 of term-finance repository, where all remaining PRs were assessed.

Note: the OpenZeppelin libraries and dependencies were excluded from the scope of this assessment.

The manual code review section of the report is focused on identifying any and all issues/vulnerabilities associated with the business logic implementation of the contract changes in scope. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout). Additionally, the manual review process focused on all known Solidity anti-patterns and attack vectors. These include, but are not limited to, the following vectors: re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers. For a more thorough, but non-exhaustive list of examined vectors, see [1, 2].

To support this review, the testing team used the following automated testing tools:

- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`
- Surya: `https://github.com/ConsenSys/surya`

Output for these automated tools is available upon request.

Findings Summary

The testing team identified a total of 4 issues during this assessment. Categorised by their severity:

- Medium: 1 issue.
- Low: 2 issues.
- Informational: 1 issue.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the code changes made to Term Finance's smart contracts, as per the scope. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as "informational".

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
TRM3-01	Incomplete Support for Multi Auction Rollovers	Medium	Resolved
TRM3-02	Lack Of Maturity Checks On Repurchase Payment	Low	Closed
TRM3-03	No Zero Address Check On Admin Wallet	Low	Resolved
TRM3-04	Miscellaneous General Comments	Informational	Open

TRM3-01	Incomplete Support for Multi Auction Rollovers		
Asset	`TermRepoRolloverManager`, `TermRepoServicer`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Low	Likelihood: High

Description

The code allows multiple auctions to be approved for rollovers, but the functionality does not appear to be fully implemented.

In `TermRepoRolloverManager`, the variable `approvedRolloverAuctions` is a mapping which allows for multiple auctions to be simultaneously approved for rollovers. It is possible to call `approveRolloverAuction()` to process multiple rollovers into multiple auctions.

However, the state variable `rolloverElections` does not allow for multiple rollover elections to different auctions to be stored at once. Also, the logic of `electRollover()`, when it checks this variable, assumes that it is always checking against a rollover election to the same target auction.

There are also no checks in `approveRolloverAuction()` to verify if `termAuction` provided as a parameter is tied to provided `auctionBidLocker`. As such, it is possible to use an arbitrary `TermAuctionBidLocker` to approve an arbitrary `TermAuction`.

Furthermore, `TermRepoServicer._getMaxRepaymentAroundRollover()` checks the value of `rolloverElections` when it calls `termRepoRolloverManager.getRolloverInstructions()` and makes calculations on this basis. If multiple auctions were approved, however, there could be rollovers that have been written over for this variable and so the calculations in `TermRepoServicer._getMaxRepaymentAroundRollover()` could be incorrect.

Recommendations

Only allow one auction to be rolled over into, or implement full multiple rollover support by including the rollover auction address as an additional level in the `rolloverElections` mapping.

Modify implementation of `approveRolloverAuction` to use `termAuction` tied to provided `auctionBidLocker`, rather than relying on supplied input.

Resolution

The finding has been resolved in PR 1136 and PR 1146.

The development team has also provided further comments on `approvedRolloverAuctions` mapping:

> *"We give the user different choices to rollover their loan, but they must select one of them. We want to support the ability for them to reconsider and choose the other option if they previously elected a roll to the undesired auction."*

TRM3-02	Lack Of Maturity Checks On Repurchase Payment		
Asset	`TermRepoServicer`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

The `submitRepurchasePayment()` function does not check if the payment is being submitted before maturity.

Previous implementation used to revert with `NotMaturedYet()` error if the current block timestamp was less than the specified `maturityTimestamp`.

Recommendations

Validate if this behaviour is as per intended business logic. If required, implement checks to ensure repurchase payments cannot be submitted prematurely, e.g.:

```
if (block.timestamp < maturityTimestamp) {
    revert NotMaturedYet();
}
```

Resolution

The finding has been closed with the following comment from the development team:

> *"We specifically removed the check on PR 1045 for the redemption timestamp because borrowers wanted the option to unlock collateral if an opportunity arose and there is no downside to the lender so long as they repay in full, including all interest due on the full term - it in fact de-risks the pool at no cost."*

TRM3-03	No Zero Address Check On Admin Wallet		
Asset	`TermController`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Medium

Description

There are no zero address checks on `controllerAdminWallet_` during `initialize()` in `TermController`.

If `controllerAdminWallet_` is set to `address(0)`, any future calls to `updateControllerAdminWallet()` will always fail due to the check on line [**151**].

Recommendations

Implement checks to ensure `controllerAdminWallet_` passed to `initialize()` is never `address(0)`.

Alternatively, modify implementation of `updateControllerAdminWallet()` to not revert on zero addresses.

Resolution

The finding has been resolved in PR 1138.

TRM3-04	Miscellaneous General Comments
Asset	`contracts/*`
Status	**Open**
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **Servicing fee collateral**
   `TermRepoRolloverManager._processRollover()` reduces the rollover bid amount by the amount of the servicing fee. One side effect of this is that, when a bid rolls over, a small amount of collateral is left in the `TermRepoLocker` of the first auction. Whilst this is not a security concern, it might cause a negative user experience.
2. **Code split between two functions**
   Perhaps as a result of the code changes, `TermAuctionBidLocker.lockRolloverBid()` has half of the code it runs split into `_lockRolloverBid()`. This split does not seem to be adding either utility or clarity, so consider combining all the code into one of the two functions.
3. **Variable name**
   Consider renaming `rolloverAuction` to `rolloverAuctionBidLocker` in both of `TermRepoRolloverElection.sol` and `TermRepoRolloverElectionSubmission.sol`, and updating any related variable names where those structs are used. It would substantially improve the clarity of the related code.
4. **Unnecessary use of upgradeable contract**
   In `TermInitializer`, the contract `AccessControlUpgradeable` is inherited from, but `TermInitializer` itself is not upgradeable. This does not cause any security issues in itself, but could be confusing. Consider using the non-upgradeable version, `AccessControl`.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The `brownie` framework was used to perform these tests and the output is given below.

```
tests/test_TermAuction.py .....                                          [  5%]
tests/test_TermAuctionBidLocker.py .....s...........                     [ 26%]
tests/test_TermAuctionOfferLocker.py ...s............                    [ 46%]
tests/test_TermController.py ...                                         [ 50%]
tests/test_TermEventEmitter.py .                                         [ 51%]
tests/test_TermInitializer.py ..                                         [ 53%]
tests/test_TermPriceConsumerV3.py .                                      [ 54%]
tests/test_TermRepoCollateralManager.py ......                           [ 61%]
tests/test_TermRepoLocker.py ...                                         [ 65%]
tests/test_TermRepoRolloverManager.py ..........                         [ 77%]
tests/test_TermRepoServicer.py .x.....                                   [ 85%]
tests/test_TermRepoToken.py .........                                    [ 96%]
tests/test_poc_repurchase.py .                                           [ 97%]
tests/test_poc_rolloverCollateral.py .                                   [ 98%]
tests/test_poc_rolloverToTwoAuctions.py .                                [100%]

=================================== warnings summary ===================================

(...snip...)

=========== 81 passed, 2 skipped, 1 xfailed, 1819 warnings in 131.31s (0:02:11) ==========
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

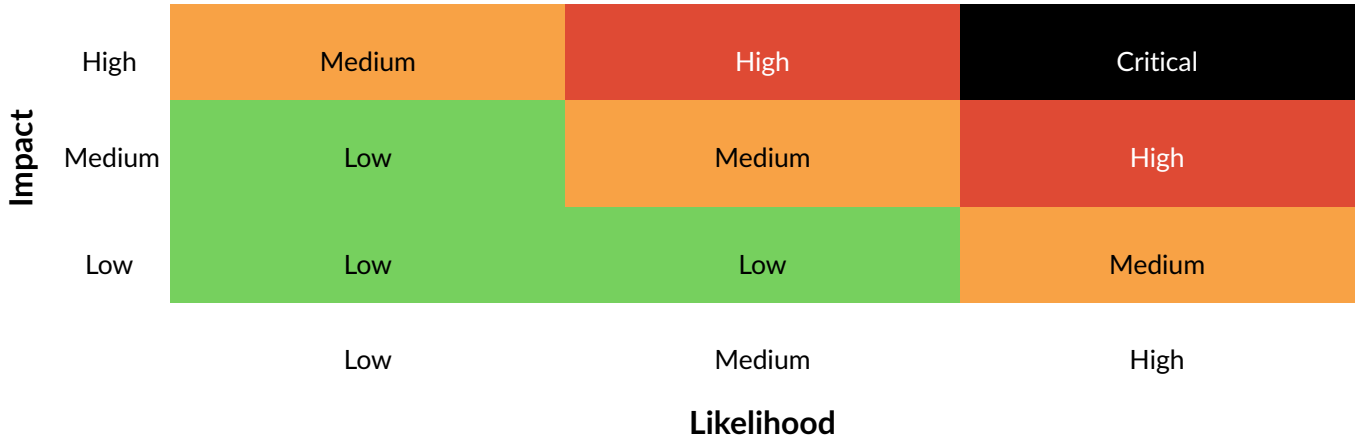

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: https://blog.sigmaprime.io/solidity-security.html. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: http://www.dasp.co/. [Accessed 2018].