Term Finance

Smart Contract Security Assessment

June 22, 2023

ABSTRACT

Dedaub was commissioned to perform a security audit of the Term Finance protocol. The auditors found that the changes-under-audit have been properly implemented and do not introduce any vulnerabilities.

BACKGROUND

The Term Finance Protocol enables noncustodial fixed-rate collateralized lending on-chain (Term Repos*) modeled on tri-party repo arrangements common in TradFi. Borrowers and lenders are matched through a unique recurring auction process (Term Auctions*) where borrowers submit sealed bids and lenders submit sealed offers that are used to determine an interest rate that clears the market for participants of that auction. Participants who bid more than the clearing rate receive loans and participants willing to lend below the clearing rate make loans, in each case at the market-clearing rate. All other participants' bids and offers are said to be "left on the table." At the conclusion of an auction, borrowers receive loan proceeds and lenders receive ERC-20 tokens (Term Repo Tokens*), which are receipts that lenders will burn to redeem for principal plus interest at maturity. Protocol smart contracts service these transactions by ledgering repayments and monitoring collateral health and liquidations.

(more information about these components can be found in the official docs of the protocol)*

SETTING & CAVEATS

This report focuses exclusively on the recent changes in the protocol. The scope of the audit included the most recent updates to the contracts of the at-the-time private repository term-finance/term-finance introduced in the following PRs:

- PR #923 · Removes authenticator logic from Term and Auction contracts
  (merged in `main` at commit `122513d36a904e42c9e2ad7fcb544bdb3652f2f6`)

- PR #931 · Emit version tags for Auctions and Terms when initialized
  (merged in `main` at commit `f96f02fe0e7e98687dc9d2e083fd3ce227918cfe`)
- PR #939 · Adds rate, min bid, servicing fee and cxl for withdrawal to emitted events
  (merged in `main` at commit `7450b28a8269664e70bb43e31a5f1a8be78657de`)

The audit focussed solely on the delta between these versions. The auditors did not re-audit the whole protocol. All the changes are fairly straightforward in nature and mainly consist of a change of authentication system and (secondarily) updates in various events. The changes are also accompanied by corresponding tests.

Previously, the protocol used an authentication system that allowed anyone to execute a transaction that could have been signed by another account, i.e. to execute on behalf of a signer. This was possible by using the `msg.data` of each call and a special struct containing the signer's information, in the signed message which gave the signer the guarantee that no one could use his signature to execute a transaction with different calldata than the one signed for. However, the current changes have removed this authentication system and replaced it with the direct use of `msg.sender`. This change does not introduce any security threats, but it removes the ability to execute a signed message on behalf of a signer, which is a design-level decision.

Two auditors worked on the codebase for 2 days on the following contracts:

```
src/
├─ Authenticator.sol
├─ TermAuction.sol
├─ TermAuctionBidLocker.sol
├─ TermAuctionOfferLocker.sol
├─ TermEventEmitter.sol
├─ TermInitializer.sol
├─ TermRepoCollateralManager.sol
├─ TermRepoRolloverManager.sol
├─ TermRepoServicer.sol
│
└─ interfaces/
    ├─ ITermAuctionBidLocker.sol
    ├─ ITermAuctionBidLockerEvents.sol
    ├─ ITermAuctionEvents.sol
    ├─ ITermAuctionOfferLocker.sol
    ├─ ITermAuctionOfferLockerEvents.sol
    ├─ ITermEventEmitter.sol
    ├─ ITermRepoCollateralManager.sol
    ├─ ITermRepoRolloverManager.sol
    ├─ ITermRepoServicer.sol
    └─ ITermRepoServicerEvents.sol
```

The audit's main target is security threats, i.e., what the community understanding would likely call "hacking", rather than the regular use of the protocol. Functional correctness (i.e., issues in "regular use") is a secondary consideration. Typically it can only be covered if we are provided with unambiguous (i.e., full-detail) specifications of what is the expected, correct behavior. In terms of functional correctness, we often trusted the code's calculations and interactions, in the absence of any other specification. Functional correctness relative to low-level calculations (including units, scaling and quantities returned from external protocols) is generally most effectively done through thorough testing rather than human auditing.

VULNERABILITIES & FUNCTIONAL ISSUES

This section details issues affecting the functionality of the contract. Dedaub generally categorizes issues according to the following severities, but may also take other considerations into account such as impact or difficulty in exploitation:

Category	Description

CRITICAL	Can be profitably exploited by any knowledgeable third-party attacker to drain a portion of the system's or users' funds OR the contract does not function as intended and severe loss of funds may result.
HIGH	Third-party attackers or faulty functionality may block the system or cause the system or users to lose funds. Important system invariants can be violated.
MEDIUM	Examples: • User or system funds can be lost when third-party systems misbehave. • DoS, under specific conditions. • Part of the functionality becomes unusable due to a programming error.
LOW	Examples: • Breaking important system invariants but without apparent consequences. • Buggy functionality for trusted users where a workaround exists. • Security issues which may manifest when the system evolves.

Issue resolution includes “dismissed” or “acknowledged” but no action taken, by the client, or “resolved”, per the auditors.

CRITICAL SEVERITY:

[No critical severity issues]

HIGH SEVERITY:

[No high severity issues]

MEDIUM SEVERITY:

[No medium severity issues]

LOW SEVERITY:

[No low severity issues]

OTHER / ADVISORY ISSUES:

This section details issues that are not thought to directly affect the functionality of the project, but we recommend considering them.

ID	Description	STATUS
A1	There are no checks to ensure the length compatibility of the arrays given in multiple functions	**INFO**

Several contracts have functions that take 2 or more arrays as arguments and iterate over them using the length of only one of them. If a length mismatch occurs and one of the arrays has fewer elements than the length used, the execution will fail because it is trying to access an out-of-bounds element. Some examples of where this happens are in the following functions:

- `TermAuction::cancelAuctionForWithdrawal():414`
- `TermAuctionBidLocker::revealBids():293`
- `TermAuctionBidLocker::auctionUnlockBid():355`
- `TermAuctionBidLocker::_unlock():521`
- `TermAuctionBidLocker::_getAllBids():574`
- `TermAuctionBidLocker::_isInInitialCollateralShortFall():795`
- `TermAuctionBidLocker::_isInMaintenanceCollateralShortFall():830`

- `TermAuctionOfferLocker::revealOffers():280`
- `TermRepoCollateralManager::journalBidCollateralToCollateralManager():706`

You could add checks before using these arrays to ensure that their lengths match, which would also save some gas in case of failure.

A2	Compiler bugs	INFO

The code is compiled with Solidity `0.8.18` or higher. For deployment, we recommend no floating pragmas, i.e., a specific version, to be confident about the baseline guarantees offered by the compiler. Version `0.8.18`, at the time of writing, has no known bugs.

DISCLAIMER

The audited contracts have been analyzed using automated techniques and extensive human inspection in accordance with state-of-the-art practices as of the date of this report. The audit makes no statements or warranties on the security of the code. On its own, it cannot be considered a sufficient assessment of the correctness of the contract. While we have conducted an analysis to the best of our ability, it is our recommendation for high-value contracts to commission several independent audits, a public bug bounty program, as well as continuous security auditing and monitoring through Dedaub Watchdog.

ABOUT DEDAUB

Dedaub offers significant security expertise combined with cutting-edge program analysis technology to secure some of the most prominent protocols in DeFi. The founders, as well as many of Dedaub's auditors, have a strong academic research background together with a real-world hacker mentality to secure code. Protocol blockchain developers hire us for our foundational analysis tools and deep expertise in program analysis, reverse engineering, DeFi exploits, cryptography and financial mathematics.