ChainLight

TERM finance

Term Finance Security Audit

：Term Finance Listing Contract

September 6, 2024

Revision 1.0

ChainLight@Theori

Table of Contents

Executive Summary

Starting on September 4, 2024, ChainLight of Theori audited the smart contracts for the Term Finance listing queue feature. These contracts implement a secondary market for Term Repo tokens, where users can list Repo tokens for sale and receive purchase tokes as their Repo tokens are purchased by other users. The conversation rate is determined by the time-adjusted clearing rate from the most recent Term Finance auction for the given Repo token, with a small contract-wide configurable markup added. The listings are stored in a queue and are filled first-come-first-serve, a design which mitigates issues identified in ChainLight's audit of the previous version of the listing contract.

During the audit, ChainLight discovered an edge case where can attacker could lock unsold listings in the contract after the term matured. Also, a design improvement was suggested to reduce the risk of gas griefing attacks due to unbounded queue iteration. Finally, ChainLight considered the impacts of auction clearing price manipulation and suggested countermeasures.

Note that the audit was conducted on the private development repository. After confirming the fixes in the private repository, ChainLight confirmed the final source code matches the code in the public repository at commit `f20a0b6d018c37612735600490c3f6e06fde39e8`.

Audit Overview

Scope

Name	Term Finance Security Audit
Target / Version	• Git Repository (term-finance-listing): commit `aa1a7c6ef29f6409f3d3b7ab467b4a92eb136591` • Final Git Repository (term-finance-listing-contracts): commit `f20a0b6d018c37612735600490c3f6e06fde39e8`
Application Type	Smart contracts
Lang. / Platforms	Smart contracts [Solidity]

Code Revision

N/A

Severity Categories

Severity	Description
Critical	The attack cost is low (not requiring much time or effort to succeed in the actual attack), and the vulnerability causes a high-impact issue. (e.g., Effect on service availability, Attacker taking financial gain)
High	An attacker can succeed in an attack which clearly causes problems in the service's operation. Even when the attack cost is high, the severity of the issue is considered "high" if the impact of the attack is remarkably high.
Medium	An attacker may perform an unintended action in the service, and the action may impact service operation. However, there are some restrictions for the actual attack to succeed.
Low	An attacker can perform an unintended action in the service, but the action does not cause significant impact or the success rate of the attack is remarkably low.
Informational	Any informational findings that do not directly impact the user or the protocol.
Note	Neutral information about the target that is not directly related to the project's safety and security.

Status Categories

Status	Description
Reported	ChainLight reported the issue to the client.
WIP	The client is working on the patch.
Patched	The client fully resolved the issue by patching the root cause.
Mitigated	The client resolved the issue by reducing the risk to an acceptable level by introducing mitigations.
Acknowledged	The client acknowledged the potential risk, but they will resolve it later.
Won't Fix	The client acknowledged the potential risk, but they decided to accept the risk.

Finding Breakdown by Severity

Category	Count	Findings
Critical	0	• N/A
High	2	• `TERM2409-001` • `TERM2409-003`
Medium	1	• `TERM2409-002`
Low	1	• `TERM2409-004`
Informational	1	• `TERM2409-005`
Note	0	• N/A

Findings

Summary

#	ID	Title	Severity	Status
1	TERM2409-001	`cancelListing` can revert if contract's tokens were redeemed	High	Patched
2	TERM2409-002	Unbounded gas usage in `_purchase` and `getTotalListings`	Medium	Mitigated
3	TERM2409-003	Auction manipulation leads to loss of funds	High	Mitigated
4	TERM2409-004	Centralization Risk	Low	Acknowledged
5	TERM2409-005	Unused `totalCost` variable in `_purchase`	Informational	Patched

#1 `TERM2409-001` `cancelListing` can revert if contract's tokens were redeemed

ID	Summary	Severity
`TERM2409-001`	Once a Term reaches maturity, anyone can redeem Repo tokens on behalf of other owners. The `cancelListing` function has logic to handle the case where somebody redeemed the listing contract's repo tokens, but it can revert due to an integer underflow.	**High**

Description

In `cancelListing`, the `removeListing` function is called before transferring the tokens to the lister:

```
 function cancelListing(uint256 listingId, bool skipRedeem) public nonReen
trant whenNotPaused {
        require(listingId < nextId, "Listing does not exist");  // Ensure
the listing exists
        Listing storage listing = listings[listingId];
        ...
        uint256 remainingAmount = listing.amount;
        address token = listing.token;
        ...
        // AUDIT NOTE: the line below will zeros the listing in storage
        removeListing(listingId);
        ...
        if (currentBalance < remainingAmount) {
            uint256 availableAmount = currentBalance;
            remainingAmount = availableAmount;
            ...
            // AUDIT NOTE: the line below can underflow because `listing.a
mount == 0`
            uint256 missingAmount = listing.amount - availableAmount;
            ...
```

```
        } else {
            // Full balance available
            require(IERC20(token).transfer(msg.sender, remainingAmount), "
Token transfer failed");
        }
```

In the normal pathway, this works because `listing.amount` is accessed indirectly through a local variable `remainingAmount`. However, in the exceptional case, `listing.amount` is accessed again after it has been cleared, leading to a possible integer underflow.

Impact

High

Immediately after the term matures, an attacker can redeem most of the Repo tokens on behalf of the contract, causing all attempts to call `cancelListing` to revert. This locks all remaining Repo tokens and their redeemed purchase tokens in the contract. Rescuing these tokens would require admin intervention.

Recommendation

Store all necessary variables from `listing` in local variables before calling `removeListing`. In this particular case, `remainingAmount` already stores `listing.amount`, so the issue is resolved by computing `missingAmount` using `remainingAmount` before it is updated:

```
        if (currentBalance < remainingAmount) {
            uint256 availableAmount = currentBalance;
            uint256 missingAmount = remainingAmount - availableAmount;
            remainingAmount = availableAmount;
            ...
        } else {
            // Full balance available
            require(IERC20(token).transfer(msg.sender, remainingAmount), "
Token transfer failed");
        }
```

References

N/A

Remediation

Patched

A variant of the recommendation was applied in commit `17f85246b7972b6d600cfd2683145cf15a07b2fe`.

#2 TERM2409-002 Unbounded gas usage in _purchase and getTotalListings

ID	Summary	Severity
TERM2409-002	A side-effect of processing listings in a queue is that iterating the queue can use significant gas. Notably, since all listings for all repo tokens are stored in the same queue, a lot of gas can be spent searching for eligible listings.	Medium

Description

In `_purchase`, the listing queue is iterated to find listings for the desired `repoToken` until the purchase order is fulfilled:

```
        while (currentListing != 0 && remainingAmount > 0) {
            Listing storage listing = listings[currentListing];

            if (listing.token == repoToken && listing.amount > 0) {
                // summarized for brevity:
                // 1. purchase max(remainingAmount, listing.amount) tokens
from the listing
                // 2. if listing is now empty, remove it from the queue an
d update currentListing
            } else {
                currentListing = listing.next;
            }
        }
```

This can use large amounts of gas for two reasons:

1. Large purchases can require many listings to fully satisfy them
2. The queue may contain many orders for other Repo tokens

Note that high gas costs may deter users from purchasing. Furthermore, if the queue becomes filled with small listings and/or listings for other Repo tokens, it may become impossible to fulfill

large orders within the block gas limit.

Impact

Medium

Unnecessarily high gas costs can deter users from making purchases (especially small purchases). Furthermore, malicious actors could spam the queue with many small listings, causing purchases for all other repo tokens to waste gas iterating the listings without making progress on their order.

Recommendation

In general, this issue is remedied by ensuring that each iteration makes significant progress towards fulfilling the order. To this end, we suggest two changes:

1. Ensure the minimum listing size is large enough (ideally a bit larger than most purchase orders)
2. Partition the queue based on Repo token. This would require tracking a separate queue state (i.e. `head`, `tail`, and optionally `nextId`) for each Repo token.

References

- https://github.com/wissalHaji/solidity-coding-advices/blob/master/best-practices/be-careful-with-loops.md

Remediation

Mitigated

The queue partitioning was implemented in commit `db5f757dbe58effdcbf8ff71054d6cacbc3c5f21`. The importance of setting an appropriate minimum listing size was also acknowledged.

#3 TERM2409-003 Auction manipulation leads to loss of funds

ID	Summary	Severity
TERM2409-003	Some repo tokens occasionally have "re-opening" auctions, where an additional auction is held after the initial auction for the token. If an attacker can cheaply manipulate the clearing price of a "re-opening" auction, the listing contract will sell previously listed tokens at an artificially low price leading to loss of funds.	High

Description

The `TermDiscountRateAdapter` contract is used to fetch the current base rate for listing sales:

```
contract TermDiscountRateAdapter is ITermDiscountRateAdapter {
    ....
    function _getAuctionRate(address repoToken) internal view returns (uin
t256) {
        (AuctionMetadata[] memory auctionMetadata, ) = TERM_CONTROLLER.get
TermAuctionResults(ITermRepoToken(repoToken).termRepoId());

        uint256 len = auctionMetadata.length;
        require(len > 0);

        return auctionMetadata[len - 1].auctionClearingRate;
    }
    ....
}
```

As shown above, it currently uses the clearing price from the latest auction for that `repoToken`. If the auction clearing prices are manipulable, an attacker can employ the following strategy:

1. Auction 1 happens with clearing price `P`
2. Lots of repo tokens get listed in the new contract. Attacker wants to buy these listed tokens at a much better price than `P`.
3. Auction 2 happens, attacker manipulates the clearing price to `P'`.

4. Now attacker can buy the listed tokens at `P'` + a small markup. Everyone who listed the tokens based on Auction 1's price loses money.

Whether it's possible to manipulate the auction price (specifically, at a lower cost than the profits from step 4) depends on many factors, such as the volume cleared in the auction and the level of auction admin vigilance. Note that specific manipulation strategies are out-of-scope for this audit, where we only focus on the impact of such manipulation on the listing contract.

Impact

High

If an attacker can manage to manipulate the clearing price of a "re-opening" auction, they can extract funds from the listers. However, we expect manipulating the auction to be nontrivial (partly due to the possibility of admin intervention). The exact feasibility of auction manipulation is outside of this audit scope.

Recommendation

As an extra security measure, we recommend adding a delay to the price reporting in the `TermDiscountRateAdapter`. The logic can be changed so that if there are multiple auction results, it doesn't immediately report the latest price. Instead, it reports the latest auction that is at least `RATE_DELAY` seconds old.

The delay should be high enough to give listers (and potentially admins) time to react and cancel their listings in the event of a manipulated auction. However, it is important that lenders in the new auction are made aware that listing immediately may result in an unexpected listing price.

References

N/A

Remediation

Mitigated

A 30-minute delay was added to the `TermDiscountRateAdapter` in commit `acfe1948ada29606e3c8b81e3d78c944895f0825`. The commit also added the ability for admins to mark specific auctions as invalid, causing the `TermDiscountRateAdapter` to skip it while finding the latest valid auction. Combined, this gives a 30 minute window for one of the following to occur in the event of a manipulated auction:

1. admins to mark a manipulated auction as invalid, preventing the `TermDiscountRateAdapter` from returning its price
2. listers to remove their listings that will be mispriced

Given the short length of this window, we strongly encourage admins to implement automation of auction invalidation based on heuristics.

#4 TERM2409-004 Centralization Risk

ID	Summary	Severity
TERM2409-004	The contract's `ADMIN_ROLE` and `DEVOPS_ROLE` are centralization risks that can lead to loss of depositor funds if a malicious actor gains control of the role.	Low

Description

Like other Term Finance contracts, the listing contract has upgradability through `UUPSUpgradeable` and `AccessControlUpgradeable`, where `_authorizeUpgrade` requires the `DEVOPS_ROLE`. Malicious contract upgrades could lead to loss of user funds. The `DEVOPS_ROLE` also can call `rescueTokens` to directly extract any token from the contract .

Similar scenarios exist with the contract's `ADMIN_ROLE`, where `setDiscountRateAdapter` can arbitrarily control the conversation rate for Repo tokens. Also, the `ADMIN_ROLE` can call `cancelListing` on any existing listing and receive all remaining repo tokens in that listing.

While upgradability and token rescue functions can aid users whose funds get stuck due to an unforeseen bug, they also pose a centralization risk. If the contract admins are compromised, a malicious actor could use these functions to steal funds from depositors.

Impact

Low

While this issue can lead to loss of user funds, it requires a malicious actor to get access to the admin keys first.

Recommendation

Ideally the contract would be non-upgradeable and not have admin-specific functions that can steal user funds. If these features are deemed necessary, the centralization risk can be mitigated by ensuring that all accounts with the `ADMIN_ROLE` is timelocked (similar to how `DEVOPS_ROLE` is already timelocked.) This gives depositors time to exit their position if they observe a pending malicious action from an admin account. However, depositors and users should be aware of the risks, as possibilities the timelock period is too short to react.

We also recommend changing `cancelListing` to transfer tokens to `listing.seller` rather than `msg.sender`, eliminating one centralization risk. Note that the `listing.seller` value will need to be cached in a local variable since the call to `removeListing` will zero out the storage slot.

References

- https://developers.term.finance/access-controls/further-details

Remediation

Acknowledged

The suggested change to `cancelListing` was implemented in commit `db5f757dbe58effdcbf8ff71054d6cacbc3c5f21`. Also, the `rescueToken` function was removed in commit `6ff756b1495ae573e9651e868d38e8decf8e0c2c`. The remaining centralization risks were acknowledged by the Term team, and there are plans to reduce admin controls in the future.

#5 TERM2409-005 Unused totalCost variable in _purchase

ID	Summary	Severity
TERM2409-005	The totalCost variable in _purchase is assigned to but never used.	Informational

Description

The totalCost variable accumulates the total amount spent to purchase the Repo tokens during a purchase order. However, the result is never used, leading to some wasted gas.

Impact

Informational

This issue has no security impact and only wastes a small amount of gas.

Recommendation

Remove the totalCost variable

References

N/A

Remediation

Patched

The totalCost variable was removed in commit b42182a61fb33552fa93da08c7d063c7b71388a2.

Revision History

Version	Date	Description
1.0	September 6, 2024	Initial version

Theori, Inc. (“We”) is acting solely for the client and is not responsible to any other party. Deliverables are valid for and should be used solely in connection with the purpose for which they were prepared as set out in our engagement agreement. You should not refer to or use our name or advice for any other purpose. The information (where appropriate) has not been verified. No representation or warranty is given as to accuracy, completeness or correctness of information in the Deliverables, any document, or any other information made available. Deliverables are for the internal use of the client and may not be used or relied upon by any person or entity other than the client. Deliverables are confidential and are not to be provided, without our authorization (preferably written), to entities or representatives of entities (including employees) that are not the client, including affiliates or representatives of affiliates of the client.