yAudit yearn v3 term vault Review

Review Resources:

- term repository and docs

Auditors:

- Fedebianu
- Panda

Table of Contents

Review Summary

Term yearn v3 vault

The Term Yearn V3 Vault is a Yearn-based vault that offers two options: users can deposit liquidity to earn yield through Yearn vaults or use their liquidity to unlock Term tokens, which would otherwise unlock later.

The contracts of the yearn-v3-term-vault Repo were reviewed over seven days. The code review was performed by two auditors between 27th August and 4th September 2024. The repository was under active development during the review, but the review was limited to commit 2272739b34f9fd5dfa3f9a2807aee9b2384acd05 for the yearn v3 term vault repo. The final commit hash is 14faba9822bd0de06f970e9b57c84f1d844c281d.

Scope

The scope of the review consisted of the following contracts at the specific commit:

```
src
├── RepoTokenList.sol
├── RepoTokenUtils.sol
├── Strategy.sol
├── TermAuctionList.sol
├── TermDiscountRateAdapter.sol
├── TermVaultEventEmitter.sol
├── periphery
│   └── StrategyAprOracle.sol
```

After the findings were presented to the yearn v3 term vault team, fixes were made and included across several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, yearn v3 term vault, and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Proper access control mechanisms are implemented, ensuring only authorized users can interact with critical functions.

Category	Mark	Description
Mathematics	Good	The mathematical calculations, including interest rates and liquidations, are handled correctly, but adjustments should be made to account for losses as outlined in the findings.
Complexity	Average	The code is well-structured but complex in handling multiple components.
Libraries	Good	The project effectively uses standard and custom libraries, following best practices in code modularity.
Decentralization	Good	The system is designed with decentralization in mind, particularly in handling term tokens and vault strategies, reducing central points of failure.
Code stability	Good	The code appears stable.
Documentation	Good	The comprehensive documentation explains the core functions and overall system architecture.
Monitoring	Good	Monitoring mechanisms are in place to track key events and changes within the system. An infrastructure to monitor the liquidations is key to keep the vault depositors safe.
Testing and verification	Average	While testing is implemented, additional focus on testing edge cases, particularly around liquidation scenarios and asset valuations, would increase confidence in the system's robustness.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas savings

    - Findings that can improve the gas efficiency of the contracts.
- Informational
    - Findings including recommendations and best practices.

Critical Findings

None

High Findings

1. High - `sellRepoToken()` doesn't take into consideration loss from liquidated assets

If the collateral is liquidated at a loss, term token holders will incur a portion of that loss. This loss can be passed on to ERC4626 strategy depositors by selling term tokens at a discounted rate in exchange for the underlying ERC20 token. The strategy does not account for potential losses, allowing term tokens to be sold at full value minus the discount rate.

Technical Details

After reviewing the `sellRepoToken()` function and its associated sub-functions, we couldn't identify any checks that ensure the term token collaterals fully cover the entire position.

In a scenario where the collateral value decreases significantly, there may be insufficient collateral to cover the full value of the position. This could lead to unexpected losses for term token holders and ERC4626 strategy depositors, as the system lacks safeguards to verify that the collateral remains adequate. Without these checks, there is a risk that term tokens could be sold even when the underlying collateral no longer fully supports the position, potentially exacerbating the financial impact.

Impact

High. Loss is carried away to the ERC4626 depositors.

Recommendation

Although tracking every loan on-chain is not feasible, monitoring them off-chain is essential. If a position loan lacks sufficient collateral, the associated term token should be automatically blacklisted. The token can be reinstated if the position's loss is recoverable. However, the risk persists if the term token has been sold before blacklisting or if the blacklist transaction is front-run.

Developer Response

https://github.com/term-finance/yearn-v3-term-vault/pull/21/files opening PR addressing this issue. We are adding a haircut to apply to underwater terms. This will be set by our oracle and applied every time present value is called

Medium Findings

1. Medium - `_totalAssetValue()` isn't considering a term token with a loss

Another issue arises when the collateral experiences a significant loss in value: the `_totalAssetValue()` function does not account for term tokens that are now undervalued due to this loss. As a result, the reported total asset value may be overstated, and the vault token value will be inflated.

Technical Details

The `getPresentValue()` function used to assess the value of the term tokens doesn't consider a potential loss. With a higher total value, vault depositors will deposit at an inflated price per share value.

Impact

Medium.

Recommendation

- Make sure `_totalAssetValue()` considers the loss from a potential liquidation of the term token collateral.
- Ensure the `totalAssets` storage value is updated when a liquidation occurs.

Developer Response

addressed https://github.com/term-finance/yearn-v3-term-vault/pull/21

Low Findings

1. Low - Incorrect `_validateAndGetOfferLocker()` validation

`_validateAndGetOfferLocker()` performs several validation steps, one of which is ensuring the auction is open. However, this validation is done wrong.

Technical Details

The require statement that ensure the auction is open in `_validateAndGetOfferLocker()` always pass considering that `offerLocker.auctionStartTime()` < `termAuction.auctionEndTime()`.

Impact

Low. The validation is passed even if the auction is not yet open or closed. However, `submitAuctionOffer()` still revert when submitting offer if the auction is not open because of the `onlyWhileAuctionOpen` modifier.

Recommendation

Rewrite the require statement:

```
        require(
-            block.timestamp > offerLocker.auctionStartTime() ||
+            block.timestamp > offerLocker.auctionStartTime() &&
                block.timestamp < termAuction.auctionEndTime(),
            "Auction not open"
        );
```

Developer Response

2. Low - Potential asset loop between Yearn vault and strategy

The strategy utilizes a Yearn vault to generate yield on idle funds. However, there is a potential risk that the same vault could theoretically incorporate this strategy, resulting in a circular asset flow.

Technical Details

Yearn multi-strategy vaults can accommodate up to 20 strategies and may choose to add this strategy to their portfolio. If the same vault that this strategy uses as a yield generator decides to incorporate this strategy, it could create an asset loop where `YEARN_VAULT -> Strategy -> YEARN_VAULT`.

Impact

Low. This scenario could lead to unexpected behavior, such as inflated or inaccurate balance reporting.

Recommendation

The documentation should include a warning about this potential scenario to prevent the vault being utilized from adding this strategy to its portfolio.

Developer Response

Acknowledged.

Gas Saving Findings

1. Gas - Avoid unnecessary loop

Technical Details

In `getPresentValue()`, iterating the whole list is unnecessary when searching for a specific token.

Impact

Gas savings.

Recommendation

Rewrite the functions to avoid loop when searching for a specific token:

```
function getPresentValue(
  RepoTokenListData storage listData,
  uint256 purchaseTokenPrecision,
  address repoTokenToMatch
) internal view returns (uint256 totalPresentValue) {
  // If the list is empty, return 0
  if (listData.head == NULL_NODE) return 0;

  if (repoTokenToMatch != address(0) && listData.discountRates[repoTokenToMatch] !=
INVALID_AUCTION_RATE) {
    totalPresentValue = _getTotalPresentValue(listData, purchaseTokenPrecision,
repoTokenToMatch);
  } else {
    address current = listData.head;
    while (current != NULL_NODE) {
      totalPresentValue += _getTotalPresentValue(listData, purchaseTokenPrecision,
current);

      // Move to the next token in the list
      current = _getNext(listData, current);
    }
  }
}

function _getTotalPresentValue(
  RepoTokenListData storage listData,
  uint256 purchaseTokenPrecision,
```

```
    address repoTokenToMatch
  ) private view returns (uint256) {
    uint256 currentMaturity = getRepoTokenMaturity(repoTokenToMatch);
    uint256 repoTokenBalance = ITermRepoToken(repoTokenToMatch).balanceOf(address(this));
    uint256 repoTokenPrecision = 10 ** ERC20(repoTokenToMatch).decimals();
    uint256 discountRate = listData.discountRates[repoTokenToMatch];

    // Convert repo token balance to base asset precision
    // (ratePrecision * repoPrecision * purchasePrecision) / (repoPrecision *
  ratePrecision) = purchasePrecision
    uint256 repoTokenBalanceInBaseAssetPrecision =
  (ITermRepoToken(repoTokenToMatch).redemptionValue() *
      repoTokenBalance *
      purchaseTokenPrecision) / (repoTokenPrecision * RepoTokenUtils.RATE_PRECISION);

    // Calculate present value based on maturity
    if (currentMaturity > block.timestamp) {
      return
        RepoTokenUtils.calculatePresentValue(repoTokenBalanceInBaseAssetPrecision,
  purchaseTokenPrecision, currentMaturity, discountRate);
    } else {
      return repoTokenBalanceInBaseAssetPrecision;
    }
  }
```

Developer Response

2. Gas - State variables can be packed into fewer storage slots

`depositLock` can be packed with an address to save a storage slot.

Technical Details

Technical Details

File: src/Strategy.sol

```
// @audit: 1 slot could be saved, by using a different order:
\*
 * struct RepoTokenListData repoTokenListData;
 * struct TermAuctionListData termAuctionListData;
 * uint256 PURCHASE_TOKEN_PRECISION; // (256 bits)
 * uint256 timeToMaturityThreshold; // (256 bits)
 * uint256 requiredReserveRatio; // (256 bits)
 * uint256 discountRateMarkup; // (256 bits)
 * uint256 repoTokenConcentrationLimit; // (256 bits)
 * mapping(address => bool) repoTokenBlacklist; // (256 bits)
 * contract ITermVaultEvents TERM_VAULT_EVENT_EMITTER; // (160 bits)
 * bool depositLock; // (8 bits)
 * contract IERC4626 YEARN_VAULT; // (160 bits)
 * contract ITermController prevTermController; // (160 bits)
 * contract ITermController currTermController; // (160 bits)
 * contract ITermDiscountRateAdapter discountRateAdapter; // (160 bits)
 */

51: ITermVaultEvents public immutable TERM_VAULT_EVENT_EMITTER;
52:     uint256 public immutable PURCHASE_TOKEN_PRECISION;
53:     IERC4626 public immutable YEARN_VAULT;
54:
55:     /// @notice State variables
56:     /// @dev Previous term controller
57:     ITermController public prevTermController;
58:     /// @dev Current term controller
59:     ITermController public currTermController;
60:     ITermDiscountRateAdapter public discountRateAdapter;
61:     RepoTokenListData internal repoTokenListData;
62:     TermAuctionListData internal termAuctionListData;
```

```
63:     uint256 public timeToMaturityThreshold; // seconds
64:     uint256 public requiredReserveRatio; // 1e18
65:     uint256 public discountRateMarkup; // 1e18 (TODO: check this)
66:     uint256 public repoTokenConcentrationLimit; // 1e18
67:     mapping(address => bool) public repoTokenBlacklist;
68:     bool public depositLock
```

src/Strategy.sol#L51

Impact

Gas savings

Recommendation

Re-order the variables

Developer Response

https://github.com/term-finance/yearn-v3-term-vault/pull/22/files#diff-c43c397aec4b970f66d861f26e4be93c66f7f376204d4cfbb9c5d0d9a44b665bR56

3. Gas - Shortcircuit rules can be used to optimize some gas usage

Some conditions may be reordered to save an SLOAD (2100 gas) or a call to an external contract. We avoid reading state variables / calling an external contract when the first part of the condition fails (with &&).

Technical Details

In these two `if` checks, the second part of the condition is variables in memory, putting them first will save some gas if the condition returns false.

```
File: src/TermAuctionList.sol

289: if (offer.termAuction.auctionCompleted() &&
repoTokenListData.discountRates[offer.repoToken] == 0) {

358: if (offer.termAuction.auctionCompleted() &&
repoTokenListData.discountRates[offer.repoToken] == 0) {
```

src/TermAuctionList.sol#L289, src/TermAuctionList.sol#L358

Impact

Gas savings.

Recommendation

Re-order the conditions.

Developer Response

https://github.com/term-finance/yearn-v3-term-vault/pull/22/files#diff-108586a4ecb3b2a244bd3a6b34a04fa9397039ad603c7741c848aa4e3e5b6099R288

4. Gas - Cache `_totalAssetValue()`

Technical Details

`_liquidReserveRatio()` is used during validations before `_validateRepoTokenConcentration()`. Both functions use `_totalAssetValue(liquidBalance)` as a calculation parameter. However, `_totalAssetValue()` is a very expensive function and it is calculated twice while it can be cached before both function calls.

Impact

Gas savings.

Recommendation

Cache `_totalAssetValue()` before `_liquidReserveRatio()` and `_validateRepoTokenConcentration()` calls. You also need to rewrite `_liquidReserveRatio()` and `liquidReserveRatio()` functions:

```
-    function _liquidReserveRatio(uint256 liquidBalance) internal view returns (uint256)
{
-        uint256 assetValue = _totalAssetValue(liquidBalance);
+    function _liquidReserveRatio(uint256 assetValue) internal view returns (uint256) {
         if (assetValue == 0) return 0;
         return liquidBalance * 1e18 / assetValue;
     }
```

```
     function liquidReserveRatio() external view returns (uint256) {
-        return _liquidReserveRatio(_totalLiquidBalance());
+        return _liquidReserveRatio(_totalAssetValue(_totalLiquidBalance()));
     }
```

Developer Response

5. Gas - Edit only `offerAmount` when editing an offer

Technical Details

When editing an offer, the whole element is overwritten. However, only the offer amount changes.

Impact

Gas savings.

Recommendation

Edit only the offer amount instead of the whole element:

```
-            termAuctionListData.offers[offerIds[0]] = PendingOffer({
-                repoToken: repoToken,
-                offerAmount: offer.amount,
-                termAuction: auction,
-                offerLocker: offerLocker
-            });
+            PendingOffer storage pendingOffer =
termAuctionListData.offers[offerIds[0]];
+            pendingOffer.offerAmount = offer.amount;
```

Developer Response

https://github.com/term-finance/yearn-v3-term-vault/pull/18 rectified

6. Gas - Optimize `_redeemRepoTokens()`

Technical Details

`_sweepAsset()` is always used before `_redeemRepoTokens()`. This way, if some repo tokens are redeemed, `YEARN_VAULT.deposit()` is called twice.

Impact

Gas savings.

Recommendation

Delete `_sweepAsset()` and rewrite `_redeemRepoTokens()` to handle all the use cases:

```
function _redeemRepoTokens(uint256 liquidAmountRequired) private {
  // Remove completed auction offers
  termAuctionListData.removeCompleted(repoTokenListData, discountRateAdapter,
address(asset));

  // Remove and redeem matured repoTokens
  repoTokenListData.removeAndRedeemMaturedTokens();

  uint256 liquidity = IERC20(asset).balanceOf(address(this));

  // Deposit excess underlying balance into Yearn Vault
  if (liquidity > liquidAmountRequired) {
    unchecked {
      YEARN_VAULT.deposit(liquidity - liquidAmountRequired, address(this));
    }
    // Withdraw shortfall from Yearn Vault to meet required liquidity
  } else if (liquidity < liquidAmountRequired) {
    unchecked {
      _withdrawAsset(liquidAmountRequired - liquidity);
    }
  }
}
```

Developer Response

rectified in https://github.com/term-finance/yearn-v3-term-vault/pull/17

7. Gas - Don’t load all offers in memory before iterations

Technical Details

`_loadOffers()` copies the offer list into a memory array before iterations in `getPresentValue()` and `getCumulativeOfferData()`. However, this operation is unnecessary and lead to an higher gas consumption than directly iterating the list, as you did for instance in `removeCompleted()`.

Impact

Gas savings.

Recommendation

Delete `_loadOffers()` and directly iterate the list instead of copying it in memory.

Developer Response

8. Gas - Use custom errors instead of `require` statements

Technical Details

The codebase contains a mix of using `require` statements and custom errors. The latter approach is the preferred option due to its lower deployment cost and runtime cost when the revert condition is met.

Impact

Gas savings.

Recommendation

Use custom errors instead of `require` statements.

Developer Response

Acknowledged.

9. Gas - Remove unused functions

Technical Details

`purchaseToRepoPrecision()` and `repoToPurchasePrecision()` are never used and can be removed.

Impact

Gas savings.

Recommendation

Remove unused functions.

Developer Response

https://github.com/term-finance/yearn-v3-term-vault/pull/22/files#diff-3b38789b3f06dba22bc6d95e8f4288d84ae6c0d35c845ff5195963957ca60577L15-L48

Informational Findings

1. Informational - Function state mutability can be restricted to pure

Pure functions don't use any storage value. It doesn't improve gas usage but has clearer schematics.

Technical Details

```
File: src/periphery/StrategyAprOracle.sol

28: function aprAfterDebtChange(
```

src/periphery/StrategyAprOracle.sol#L28

Impact

Informational

Recommendation

Change the visibility modifier.

Developer Response

https://github.com/term-finance/yearn-v3-term-vault/pull/22

2. Informational - Unnecessary cast

The variable is being cast to its own type

Technical Details

```
File: src/RepoTokenList.sol

327: if (purchaseToken != address(asset)) {
```

src/RepoTokenList.sol#L327

```
File: src/Strategy.sol

347: revert RepoTokenList.InvalidRepoToken(address(repoToken));

746: revert RepoTokenList.InvalidRepoToken(address(repoToken));

1008: revert RepoTokenList.InvalidRepoToken(address(repoToken));
```

src/Strategy.sol#L347, src/Strategy.sol#L746, src/Strategy.sol#L1008

Impact

Informational

Recommendation

Remove the unnecessary casting.

Developer Response

https://github.com/term-finance/yearn-v3-term-vault/pull/22

3. Informational - Emit vault address in `TermVaultEventEmitter`

The `TermVaultEventEmitter` can be used by multiple vaults, but the emitted events currently do not include the corresponding vault address. Including the vault address in the emitted events could simplify event tracking.

Technical Details

The `TermVaultEventEmitter` allows multiple vaults to hold the `VAULT_CONTRACT` role. However, when events are emitted, the vault address is not included, which complicates tracking these events.

Impact

Informatrional

Recommendation

Add the vault address if more than one Strategy is to be deployed.

Developer Response

https://github.com/term-finance/yearn-v3-term-vault/pull/22

4. Informational - Initial values aren't set

Some variables do not have their initial values set and will require a call to their respective setter functions immediately after the constructor is invoked.

Technical Details

The following variables are not initialized by the contract's constructor and will default to zero: timeToMaturityThreshold, requiredReserveRatio, discountRateMarkup, and repoTokenConcentrationLimit. While setter functions are available to assign values to these variables, it is generally recommended to set initial values directly in the constructor.

Impact

Informational

Recommendation

Initialize these variables directly in the constructor to ensure they have non-zero values from the outset.

Developer Response

https://github.com/term-finance/yearn-v3-term-vault/pull/22

5. Informational - `TODO` left in the code

Technical Details

```
File: src/Strategy.sol

65: uint256 public discountRateMarkup; // 1e18 (TODO: check this)

1244: TODO: If desired Implement deposit limit logic and any needed state variables .

1307: TODO: If desired implement simple logic to free deployed funds.
```

src/Strategy.sol#L65, src/Strategy.sol#L1244, src/Strategy.sol#L1307

```
File: src/TermAuctionList.sol

229: // TODO: do we need to validate termDeployed(repoToken) here?
```

src/TermAuctionList.sol#L229

```
File: src/periphery/StrategyAprOracle.sol

32: // TODO: Implement any necessary logic to return the most accurate
```

src/periphery/StrategyAprOracle.sol#L32

Impact

Informational

Recommendation

Make sure all of the todos are done

Developer Response

https://github.com/term-finance/yearn-v3-term-vault/pull/22

6. Informational - On auction update deposit to vault when funds are freed

Funds can be freed from the `TermAuctionOfferLocker`, they should be deposited into the vault and not stay idle in the contract.

Technical Details

`submitAuctionOffer()` can update an existing auction, if the updated auction has fewer tokens than the previous one, assets will be returned to the strategy, they can be deposited to the vault to benefit from the vault yield.

Impact

Informational

Recommendation

Handle the case where funds are returned and can be deposited.

Developer Response

https://github.com/term-finance/yearn-v3-term-vault/pull/24/files

7. Informational - Typos

Typos found on the code.

Technical Details

```
File: src/TermAuctionList.sol


// @audit: ammount should be amount
213: offer.offerLocker.unlockOffers(offerIds); // unlocking offer in this scenario
withdraws offer ammount
```

src/TermAuctionList.sol#L213

Impact

Informational.

Recommendation

Fix typos

Developer Response

https://github.com/term-finance/yearn-v3-term-vault/pull/22

8. Informational - Unnecessary else block

An empty else block should be removed.

Technical Details

```
File: TermAuctionList.sol
204:  } else {
205:      // Otherwise, do nothing if the offer is still pending
206:  }
```

TermAuctionList.sol#L204

Impact

Informational

Recommendation

Remove empty else block.

Developer Response

Acknowledged.

9. Informational - Incorrect NatSpec

Technical Details

`auctionClosed()` NatSpec is incorrect. The function does not close an auction, but it should be called when an auction is closed to handle funds.

Impact

Informational.

Recommendation

Correct the function NatSpec.

Developer Response

https://github.com/term-finance/yearn-v3-term-vault/pull/22/files#diff-c43c397aec4b970f66d861f26e4be93c66f7f376204d4cfbb9c5d0d9a44b665bR979

Final remarks

This audit provides a comprehensive review of the system, identifying key areas where improvements are needed to enhance security, performance, and maintainability. While some critical issues must be addressed promptly to mitigate risks, the overall system demonstrates a solid foundation with a strong potential for optimization.