yAudit Term yearn v3 vault Review

Review Resources:

- Code repository and docs

Auditors:

- Panda
- Fedebianu

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 Term yearn v3 vault Repo were reviewed over three days. The code review was performed by two auditors between 17th and 20th December 2024. The repository was under active development during the review, but the review was limited to the latest commit 99cd2b720c7b7c8c498d6238504b42315e06a8c2 for the Term yearn v3 vault repo. The review focused particularly on changes made since the previous audit at commit 14faba9822bd0de06f970e9b57c84f1d844c281d. The final commit after review is c6f202507b2c2f966ba71e128f885031080c343a.

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
```

After the findings were presented to the Term yearn v3 vault team, fixes were made and included in 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, Term yearn v3 vault and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	The contracts implement proper access control mechanisms with clear role separation and well-defined permissions.
Mathematics	Good	No complex calculations that could lead to precision issues.
Complexity	Good	The codebase maintains a reasonable level of complexity with clear separation of concerns and modular design.
Libraries	Good	The project uses well-tested and standard libraries, including OpenZeppelin contracts, and implements them correctly.
Decentralization	Good	The system maintains appropriate levels of decentralization with minimal privileged operations.
Code stability	Good	The codebase is stable with well-structured code and consistent patterns throughout. Changes from the previous audit were focused and purposeful.
Documentation	Good	Code is well-documented with clear comments explaining functionality and important considerations.

Category	Mark	Description
Monitoring	Good	Appropriate events are emitted for key operations, allowing for effective monitoring of contract activity.
Testing and verification	Good	Comprehensive test suite covering core functionality and edge cases. Good test coverage across the codebase.

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 - Return value shadowing in `_validateAuctionExistence()` breaks auction rate invalidation mechanism

`_validateAuctionExistence()` in `TermDiscountRateAdapter.sol` contains a critical implementation error that causes it always to return `false`, regardless of whether the auction exists or not. This makes it impossible to invalidate any auction rate through `setAuctionRateValidator()`.

Technical Details

`_validateAuctionExistence()` declares a named return variable `auctionExists` but then shadows it with a local variable of the same name. This breaks `setAuctionRateValidator()` functionality because `auctionExists` will always be false due to the above bug.

Since `setAuctionRateValidator()` will always revert, marking any auction rate as invalid is impossible, even when necessary. This is particularly severe because `_getDiscountRate()` relies on `rateInvalid` to determine which auction rate to use for pricing. Without the ability to invalidate rates, the system must use potentially compromised or incorrect auction rates, as there's no way to exclude them from the pricing calculation. This could lead to incorrect pricing and valuation throughout the system, potentially affecting all operations that rely on these rates.

Impact

High. The auction rate invalidation mechanism is completely broken.

Recommendation

Correct the `_validateAuctionExistence()` implementation:

```
    function _validateAuctionExistence(AuctionMetadata[] memory auctionMetadata,
bytes32 termAuctionId) private view returns(bool auctionExists) {
        // Check if the termAuctionId exists in the metadata
-       bool auctionExists;
        for (uint256 i = 0; i < auctionMetadata.length; i++) {
            if (auctionMetadata[i].termAuctionId == termAuctionId) {
                auctionExists = true;
                break;
            }
        }
    }
```

This can be seen as part of the compiler warnings; activate; `deny_warnings` as part of your makefile to stop compilation on warnings and prevent such errors from happening again.

Developer Response

Fix addressed in this PR (https://github.com/term-finance/yearn-v3-term-vault-contracts/pull/6) and also changed function visibility to pure as recommended by the compiler.

Medium Findings

None

Low Findings

None

Gas Saving Findings

1. Gas - Cache storage variables

Technical Details

In `Strategy.sol`, multiple functions unnecessarily read `strategyState` repeatedly from storage, resulting in higher gas costs.

Impact

Gas savings.

Recommendation

Cache storage variables.

Developer Response

Acknowledge. Won’t fix.

2. Gas - Remove unused variables

Technical Details

The following variables were created but never used.

```
File: Strategy.sol
576:         ITermController prevTermController = strategyState.prevTermController;
577:         ITermController currTermController = strategyState.currTermController;
578:
```

Strategy.sol#L576-L577

Impact

Gas savings.

Recommendation

Remove the variables.

Developer Response

Fixed as part of PR#7

3. Gas - Unnecessary variable creation

Technical Details

The variable is not used and can be directly assigned at line 192.

```
File: Strategy.sol
178:         ITermController newTermController = ITermController(newTermControllerAddr);
```

Strategy.sol#L178-L178

Impact

Gas savings.

Recommendation

```
File: Strategy.sol
-178:         ITermController newTermController =
ITermController(newTermControllerAddr);
-192:         strategyState.currTermController = newTermController;
+192:         strategyState.currTermController = ITermController(newTermControllerAddr);
```

Developer Response

Fixed as part of PR#7

Informational Findings

1. Informational - Remove unnecessary return parameter in `validateAndInsertRepoToken()`

Technical Details

`validateAndInsertRepoToken()` returns a `discountRate` parameter that is never used by any calling functions.

Impact

Informational.

Recommendation

Remove the unused `discountRate` return parameter.

```
  function validateAndInsertRepoToken(
      RepoTokenListData storage listData,
      ITermRepoToken repoToken,
      ITermDiscountRateAdapter discountRateAdapter,
      address asset
-   ) internal returns (bool validRepoToken, uint256 discountRate, uint256 
redemptionTimestamp) {
+   ) internal returns (bool validRepoToken, uint256 redemptionTimestamp) {
-       discountRate = listData.discountRates[address(repoToken)];
+       uint256 discountRate = listData.discountRates[address(repoToken)];
        if (discountRate != INVALID_AUCTION_RATE) {
            (redemptionTimestamp,,,) = repoToken.config();

            // skip matured repoTokens
            if (redemptionTimestamp < block.timestamp) {
-               return (false, discountRate, redemptionTimestamp);
+               return (false, redemptionTimestamp);
            }

            uint256 oracleRate;
            try discountRateAdapter.getDiscountRate(address(repoToken)) returns (uint256 
rate) {
                oracleRate = rate;
            } catch {}

            if (oracleRate != 0) {
                if (discountRate != oracleRate) {
                    listData.discountRates[address(repoToken)] = oracleRate;
                }
            }
        } else {
            try discountRateAdapter.getDiscountRate(address(repoToken)) returns (uint256
rate) {
                discountRate = rate == 0 ? ZERO_AUCTION_RATE : rate;
            } catch {
                discountRate = INVALID_AUCTION_RATE;
-               return (false, discountRate, redemptionTimestamp);
+               return (false, redemptionTimestamp);
            }

            bool isRepoTokenValid;

            (isRepoTokenValid, redemptionTimestamp) = validateRepoToken(listData,
repoToken, asset);
            if (!isRepoTokenValid) {
-               return (false, discountRate, redemptionTimestamp);
+               return (false, redemptionTimestamp);
            }
            insertSorted(listData, address(repoToken));
            listData.discountRates[address(repoToken)] = discountRate;
        }

-       return (true, discountRate, redemptionTimestamp);
+       return (true, redemptionTimestamp);
    }
```

Developer Response

addressed https://github.com/term-finance/yearn-v3-term-vault-contracts/pull/7

Final remarks

The review examined the codebase changes implemented following Runtime Verification's formal verification audit. Overall, the fixes from the previous audit were successfully implemented, and the codebase shows good maturity. During this review:

- A critical variable shadowing bug was identified in the auction validation mechanism
- Several gas optimization opportunities were discovered
- Additional code quality improvements were suggested

While the codebase is generally well-structured and secure, we recommend addressing the high-severity finding related to auction rate validation as soon as possible.
