yAudit yearn v3 Review

Review Resources:

- Two different codebases from two different repositories.

Auditors:

- adriro
- panda

Table of Contents

Review Summary

Yearn V3

The Yearn V3 Vaults project includes an ERC4626 compliant Vault contract that handles various functionalities such as deposits, withdrawals, strategy management, and profit reporting. Additionally, the tokenized-strategy contract provides a single yield source ERC4626 compliant token.

The codebases for the Yearn V3 Vaults and tokenized-strategy can be found in the

following repositories:

- Yearn V3 Vaults: yearn-vaults-v3
- Tokenized Strategy: tokenized-strategy

The code review of the Yearn V3 Vaults and tokenized-strategy repositories occurred between July 3rd and July 28th, 2023. The review period lasted for 25 days, involving two auditors. The repositories were actively developed during this time, and the review specifically focused on the latest commit at the start of the review, denoted as version v3.0.1-beta for the vault codebase and v3.0.1-beta for the tokenized-strategy.

Scope

The scope of the review consisted of the following contracts in two different repositories:

yearn-vaults-v3 tag v3.0.1-beta

```
├── VaultFactory.vy
├── VaultV3.vy
```

tokenized-strategy tag v3.0.1-beta

```
├── BaseTokenizedStrategy.sol
├── TokenizedStrategy.sol
```

After the findings were presented to the Yearn Finance 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, Yearn Finance and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	The access control mechanisms implemented in the codebase are well-designed and effectively manage permissions. However, there is a need to impose restrictions on the vault role opening feature to enhance security.
Mathematics	Good	The usage of unsafe math in the codebase is carefully implemented and takes into account the default safe math features provided by Vyper and Solidity versions 0.8 and above.
Complexity	Good	The design of the Vault contract is impressive, showcasing a thoughtful approach. The tokenized strategy proxy design is quite uncommon, setting it apart from conventional practices. The process of withdrawing from the Vault involves intricacies and can be perceived as complex.
Libraries	Good	Usage of standard openzeppelin libraries
Decentralization	Average	Many of the vault functionalities are safeguarded, and the administrators hold responsibility for reporting strategy harvests and funds allocation.
Code stability	Average	The codebase for the vault is completed and considered final, while the tokenized strategy serves as a foundational component for a strategy that will need to undergo additional audits.
Documentation	Average	While the documentation for the vault Vyper is excellent, the TokenizedStrategy codebase suffers from a lack of NatSpec documentation on multiple functions, despite being well-documented in some areas.

Category	Mark	Description
Monitoring	Good	Functions in the codebase emit events, ensuring that important occurrences or state changes are recorded and observable.
Testing and verification	Average	Expanding the code coverage by integrating actual strategies is an essential step that needs to be taken.

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

Medium Findings

1. Medium - Vault should mint exactly the specified number of shares

The `mint()` function present in the VaultV3.vy contract does a double rounding which may end up minting a different number of shares.

Technical Details

Shares are minted by first converting the `shares` argument to `assets` using the `_convert_to_assets()` function with the option to round up the calculation. The `assets` variable is then forwarded to the internal function `_deposit()`, which takes this amount and recalculates the number of shares in the `_issue_shares_for_amount()` function.

This breaks the compliance with EIP-4626, as the `mint()` function is expected to mint the receiver the exact number of shares:

> Mints exactly `shares` Vault shares to `receiver` by depositing `assets` of underlying tokens.

Impact
Medium. The caller expects the exact number of shares to be minted, in accordance with the EIP-4626 standard.

Recommendation
The implementation should mint the specified number of shares in the `shares` parameter.

Developer Response
Added a new internal '_mint' function that is used now instead of during mints. Changes can be found here.

2. Medium - Non-compliance with ERC4626 Withdrawal Standard
In order to comply with the ERC4626 standard, the `withdraw()` method should `burn` shares and send the exact number of `assets` to the `receiver`.

Technical Details
See: EIP-4626

The methods used in both the v3 vault and the v3 strategy are to convert `assets` into `shares` and to burn this number of `shares`. If any losses occur, the number of `assets` to be sent back is currently reduced. However, this approach does not align with the ERC4626 protocol. Instead, more shares should be burned to match the requested number of `assets`.

https://github.com/yearn/yearn-vaults-v3/blob
/3e51c08d88d2764b39348c98f20e18c51b475722/contracts/VaultV3.vy#L1523
https://github.com/yearn/tokenized-strategy
/blob/72a71b16088f1a7821f299aa013d824b3dbb8d4e
/src/TokenizedStrategy.sol#L464

Impact
Medium. Not complying with ERC4626 may have side effects on the integration of the protocol.

Recommendation

To address this issue, it is recommended to modify the withdrawal process to take into account the requested number of `assets`.

Developer Response

For both the vault and the strategy we have added an optional “max loss” variable that can be added to the standard 4626 withdraw or redeem calls. This variable will default to 0 for all withdraw calls meaning it will need to be manually increased in order to allow for any realized losses to be passed on during the withdrawal. The variable will default to 100% on all redeem calls. Meaning the normal function of losses should not change at all when using redeem but users can use this extra variable to limit any loss.

This also means we have updated the vault’s internal _redeem to pull funds from a strategy using redeem instead of withdraw to allow for losses natively from Yearn strategies without breaking 4626 standard integration.

Changes can be found here.

Low Findings

1. Low - `set_open_role()` might only be used for a single role while putting at risk all the others

The `set_open_role()` is a method that enables admins to open a role to everyone. While this method can be useful in certain circumstances, it carries a high degree of risk if not used correctly. A mistake, such as opening a role that should remain closed, could potentially result in irreversible and damaging effects on the entire vault.

The current role management system design does not include a confirmation or second check mechanism, increasing the risk of errors. In addition, the majority of the roles in the system are not intended to be open. In particular, only the **REPORTING_MANAGER** role might need to be opened under certain circumstances.

Technical Details

VaultV3.vy#L1255-L1275

Impact

Low.

Recommendation

To mitigate this risk, it's recommended to remove the `set_open_role` function. Instead, a more restrictive function could be introduced. This new function would only permit specific roles to be opened, such as **REPORTING_MANAGER**. This approach would preserve the flexibility of the system while significantly reducing the risk of catastrophic errors.

Developer Response

Risk accepted.

We would like the vault to be as customizable as possible without us making any of the decisions on which should or should not be open. The risk due to a mistake is also mitigated since no one role can steal any funds from the vault and any mistakenly open role can be closed.

2. Low - `lastReport()` function does not return timestamp of last report for all cases

The `lastReport()` function present in the VaultV3.vy contract is not technically correct as the underlying timestamp is not updated in all cases.

Technical Details

The implementation of `lastReport()` returns the value of the `last_profit_update` storage variable.

This timestamp is handled during the call to `process_report()` but is only updated if shares are burned (line 425) or if there are locked shares after logic is executed (line 1139).

Impact

Low. This is a view function that is not directly used in the protocol contracts.

Recommendation

Rename the function and its documentation so it is aligned with the current behavior, or always update the `last_profit_update` variable while processing reports.

Developer Response

Updated the function name and description to properly signal its purpose.

Changes can be found here

3. Low - Use a two step process to update the management account

The `management` account present in the TokenizedStrategy.sol contract may be accidentally transferred to the wrong account.

Technical Details

The implementation of the `setManagement()` function performs the role transfer in a single step, by directly transferring this key role to the specified account.

Impact

Low. Management may be accidentally transferred to the wrong account. Requires user mistake.

Recommendation

Similar to how the role manager account is transferred in the VaultV3.vy contract, implement a two-step process to update the management account in strategies.

Developer Response

Two-step process added.

Changes can be found here

4. Low - Strategy deposit limits should be checked on the receiver account

According to the EIP-4626 spec, deposit limits are expected to be defined in terms of the `receiver` account, not the caller.

Technical Details

The implementation of the `deposit()` and `mint()` functions check for deposit limits on the caller (`msg.sender`) account.

The EIP-4626 standard specifies that these limits should be queried for the `receiver` account. Taking the `maxDeposit` function as an example, the specification states:

> Maximum amount of the underlying asset that can be deposited into the Vault for the `receiver`, through a `deposit` call.

Impact

Low.

Recommendation

The `_deposit()` function should check the limits on the `receiver` account.

```
require(
    assets <=
        IBaseTokenizedStrategy(address(this)).availableDepositLimit(
            receiver
        ),
    "ERC4626: deposit more than max"
);
```

Additionally, consider reusing the `maxDeposit()` function (i.e. `require(assets <= maxDeposit(receiver))`).

Developer Response

Updated to use receiver.

The available deposit limit is manually checked in order to avoid a second unnecessary check if the vault is shut down.

Changes can be found here

5. Low - Strategy may allow to redeem more assets than the available limit

The `maxRedeem()` function rounds up the calculated shares, which may allow to withdraw more assets than specified by `availableWithdrawLimit()`.

Technical Details

In order to calculate the maximum number of shares that can be minted, the implementation of `maxRedeem()` takes the amount of assets specified by `availableWithdrawLimit()` and calculates the associated number of shares by using `previewWithdraw()`, which uses up rounding.

The resulting number of shares can then be used in `redeem()` to withdraw assets according to the calculation of `previewRedeem()`.

This double rounding may, under certain circumstances, result in more assets than the original limit specified by `availableWithdrawLimit()`.

Impact

Low. An account can redeem the maximum number of shares which may end up totalling more assets than the available limit due to rounding issues.

Recommendation

Round down the calculation present in line 646.

Developer Response

We don't believe this is an issue.

6. Low - Inconsistent rounding of `losses_user_share`

The `losses_user_share` variable should be rounded up.

Technical Details

```
    losses_user_share: uint256 = assets_needed - assets_needed * strategy_assets / strategy_current_debt
```

The calculation is done with a loss of precision, it should be rounded up instead of down.

yearn

Impact

Low. Losses of shares are lowered.

Recommendation

Round up the `losses_user_share` calculation.

```
    losses_user_share: uint256 = assets_needed - (assets_needed * strategy_assets / strategy_current_debt +1)
```

Developer Response

Updated to round up.

Changes can be found here

7. Low - Potential fund loss due to missing zero address check

In the function `_redeem()` of the Yearn Finance contract (https://github.com/yearn), there is no check for whether the receiver address parameter is the zero address (0x0).

Technical Details

The lack of a zero address check on the receiver could lead to unintentional loss of

funds. If the receiver address were mistakenly set to the zero address, the transferred funds would be irretrievably lost. Given that this function appears to handle fund withdrawal, this could potentially lead to a significant loss.

https://github.com/yearn/yearn-vaults-v3/blob/3e51c08d88d2764b39348c98f20e18c51b475722/contracts/VaultV3.vy#L802

Impact

Low. Funds can be lost if the user makes a mistake.

Recommendation

```
assert receiver != empty(address), "Receiver address cannot be the zero address"
```

Developer Response

Extra checks were added on both the vault and the strategy.

Changes can be found here and here.

8. Low - Update debt does not check that strategy is active

VaultV3.vy fails to validate that the `strategy` argument in an active strategy.

Technical Details

The `update_debt()` function does not check that the given `strategy` is currently active (i.e. `self.strategies[strategy].activation != 0`).

Impact

Low. Debt can't be decreased (as `current_debt == 0`) or increased (since `max_debt == 0`).

Recommendation

Check that the given strategy is active.

```
@external
@nonreentrant("lock")
def update_debt(strategy: address, target_debt: uint256) -> uint256:
    """
    @notice Update the debt for a strategy.
    @param strategy The strategy to update the debt for.
    @param target_debt The target debt for the strategy.
    @return The amount of debt added or removed.
```

```
    """
    self._enforce_role(msg.sender, Roles.DEBT_MANAGER)
    assert self.strategies[strategy].activation != 0, "inactive strategy"
    return self._update_debt(strategy, target_debt)
```

Developer Response

Accepted. There is no risk of funds being incorrectly sent to non-added strategies.

Gas Findings

1. Gas - Cache storage variable locally to prevent multiple reads from storage

Cache variables read from storage to prevent multiple SLOAD operations.

Technical Details

- VaultV3.vy#L328
- VaultV3.vy#L334
- VaultV3.vy#L838
- VaultV3.vy#L1329

Impact

Gas savings.

Recommendation

Cache state in local variables instead of reading again from storage.

Developer Response

Included for the allowances and `availableDepositLimit`.

Did not include for the `_revoke_strategy` as the expected behavior is to not be force revoking and should only be read once.

Changes can be found here

2. Gas - Don’t initialize variables with default value

Avoid initializing variables with its default value to save gas.

Technical Details

- VaultV3.vy#L294

Impact
Gas savings.

Recommendation
Remove the initialization.

Developer Response
Changed.

Changes can be found here

3. Gas - Function can be marked external
The function can be marked external when not used in the function. The EIP4626 doesn't specify the visibility of the functions, using external will still comply with the EIP.

Technical Details
on the `TokenizedStrategy.sol` contract, several functions can be marked external.

TokenizedStrategy.sol#L587 TokenizedStrategy.sol#L599
TokenizedStrategy.sol#L615 TokenizedStrategy.sol#L1355
TokenizedStrategy.sol#L1404 TokenizedStrategy.sol#L1450
TokenizedStrategy.sol#L1482 TokenizedStrategy.sol#L1509
TokenizedStrategy.sol#L1536

Impact
Gas savings.

Recommendation
Update the function visibility

Developer Response
Changed.

Changes can be found here

4. Gas - Use != 0 instead of > 0 for comparison
To save gas use != 0 instead of > 0 for integer comparison.

Technical Details
TokenizedStrategy.sol#L878 TokenizedStrategy.sol#L885
TokenizedStrategy.sol#L897 TokenizedStrategy.sol#L911

TokenizedStrategy.sol#L915 TokenizedStrategy.sol#L925
TokenizedStrategy.sol#L934 TokenizedStrategy.sol#L942
TokenizedStrategy.sol#L1340

Impact

Gas savings.

Recommendation

Use not equal to instead of greater than.

Developer Response

Changed.

Changes can be found here.

5. Gas - Use unchecked for subtractions where the operands cannot underflow

The contract uses `unchecked` several times to save on gas, but some safe operations were missed.

Technical Details

TokenizedStrategy.sol#L1065

TokenizedStrategy.sol#L1073

VaultV3.vy#L304

VaultV3.vy#L309

VaultV3.vy#L310

VaultV3.vy#L378

VaultV3.vy#L533

VaultV3.vy#L581

VaultV3.vy#L695

VaultV3.vy#L889

VaultV3.vy#L897

VaultV3.vy#L950

VaultV3.vy#L1025

VaultV3.vy#L1028

VaultV3.vy#L1329

Impact
Gas savings.

Recommendation
Use unchecked for the substractions.

Developer Response
Added.

Changes can be found here and here

6. Gas - Vyper 0.3.8 introduces transient storage for non-reentrancy checks, leading to notable gas savings

The most recent Vyper update employs tload/tstore for nonreentrancy, making it a more gas-efficient option.

Technical Details
yearn vault

Impact
Gas savings.

Recommendation
Use vyper 0.3.9.

Developer Response
Accepted. The contracts are built to work on any EVM chain which would be negated if we added the new Ethereum only opcodes. We also want the bytecode to be the same for every version so we can use create2 to get the same address on each chain we launch on.

Informational Findings

1. Informational - Unnecessary return variable declaration

A declared return variable isn't used.

Technical Details

BaseTokenizedStrategy.sol

Impact

Gas saving.

Recommendation

Consider changing the variable to be an unnamed one, since the variable is never assigned, nor is it returned by name. If the optimizer is not turned on, leaving the code as it is will also waste gas for the stack variable.

Developer Response

Removed.

Changes can be found here

2. Informational - Use constants for literal or magic values

Consider defining constants for literal or magic values as it improves readability and prevents duplication of config values.

Technical Details

- TokenizedStrategy.sol#L1302
- TokenizedStrategy.sol#L1302

Impact

Informational.

Recommendation

Define values as constants.

Developer Response

Added.

Changes can be found here

3. Informational - Prefer `abi.encodeCall()` over `abi.encodeWithSignature()` or `abi.encodeWithSelector()`

Prefer using `abi.encodeCall()` as this variant does a type check of the arguments.

Technical Details

- BaseTokenizedStrategy.sol#L433

Impact
Informational.

Recommendation
Replace `abi.encodeWithSignature()` or `abi.encodeWithSelector()` with `abi.encodeCall()`.

Developer Response
Changed.

Changes can be found here

4. Informational - Optional ERC-20 functions
The `name()`, `symbol()` and `decimals()` functions of ERC-20 tokens are optional.

Technical Details
- VaultV3.vy#L280
- TokenizedStrategy.sol#L383
- TokenizedStrategy.sol#L1366

Impact
Informational.

Recommendation
If contracts are expected to operate with such rare cases, provide a default implementation in case those functions are not present.

Developer Response
Accepted. Contracts should not be used for tokens without those attributes.

5. Informational - Typos in tokenized-strategy repo
Several typos were found in the tokenized-strategy codebase:

withen, mangement, unlcokdedShares, intialize, defualt, permsionless, sandwhiched, attemppt, puroposes, adress, avialable, neccesary.

Most were fixed as part of f315185e62b63f718857a1e0edbf961a18866ac5 but one remain on **unlcokdedShares**.

Impact
Informational.

Recommendation

Fix the typo.

Developer Response

Fixed.

Changes can be found here

6. Informational - Vyper `raw_call()` can be replaced with the new vyper default_return syntax

The latest version, vyper 0.3.4, introduces a new syntax that allows the addition of a default_return_value to a call.

Technical Details

The existing codebase uses low-level `raw_call` to handle non-standard ERC20 calls. This can be simplified using the new syntax.

Impact

Informational. The new syntax will enhance code readability.

Recommendation

Consider employing the new syntax:

```
    response: Bytes[32] = raw_call(
        token,
        concat(
            method_id("transferFrom(address,address,uint256)"),
            convert(sender, bytes32),
            convert(receiver, bytes32),
            convert(amount, bytes32),
        ),
        max_outsize=32,
    )
    if len(response) > 0:
        assert convert(response, bool), "Transfer failed!"
```

This can be replaced by:

```
assert ERC20(token).transfer(receiver, total_claimable, default_return_value=True)
```

Developer Response

Updated.

Changes can be found here

7. Informational - Selector clashing attack in strategies

A bad actor may hide a malicious function in the strategy implementation that gets triggered when calling one of the functions in the TokenizedStrategy.sol contract.

Technical Details

The current design of strategies is composed of two contracts, an abstract BaseTokenizedStrategy.sol contract inherited by strategist and a TokenizedStrategy.sol contract that implements the vault's core logic.

Communication between the two is done using `delegatecall`. Similar to the proxy pattern, the main strategy contract uses a `fallback()` function to delegatecall any non-matching function to the TokenizedStrategy.sol contract.

Since function selectors are 4 bytes long, it is feasible to mine a function signature that matches one of the functions in the TokenizedStrategy.sol contract. A bad actor can then hide such a function in the strategy implementation that will be unintentionally called while trying to access the "real" function. Due to the current contract design, this function will be caught by the strategy implementation instead of being delegated to the TokenizedStrategy.sol contract, as function dispatch will happen first in the main strategy contract.

Impact

Informational. Strategies are expected to be reviewed.

Recommendation

While reviewing strategies, make sure there are no matching selectors between the functions present in the TokenizedStrategy.sol contract and the functions present in the strategy implementation contract.

Developer Response

Known. Will check during development and testing.

8. Informational - Lack of event logging in `tend()`

In the `tend()` function in the contract (link to the contract repo), there's a lack of event logging upon the completion of the operation.

Technical Details

TokenizedStrategy.sol#L1051

Impact

Informational.

Recommendation

Add an event

```
emit Tend(beforeBalance, afterBalance, S.totalIdle, S.totalDebt);
```

Developer Response

Accepted.

9. Informational - Lack of event logging in `emergencyWithdraw()`

In the `emergencyWithdraw()` function in the contract, there's a lack of event logging upon the execution of the emergency withdrawal process.

Technical Details

TokenizedStrategy.sol#L1122

Impact

Informational.

Recommendation

Add an event to the `emergencyWithdraw()` function.

Developer Response

Accepted.

Final Remark

The codebase for Yearn V3 Vaults and tokenized-strategy was reviewed between July 3rd and July 28th, 2023. The review specifically focused on the latest commit at the start of the review.

Overall, the codebase showed good quality and a thoughtful approach towards smart contract design. Some areas, like the tokenized strategy proxy design, were quite unique and showcased an innovative approach towards on chain gas deployment limitations. The tokenized strategy contract also provides a single yield source ERC4626 compliant token, which adds additional functionality for users.

The main areas of concern raised by the review were regarding compliance with

ERC4626 standards. In the `mint()` function and the `withdraw()` method, there were instances where the number of shares minted or burned did not exactly match the specified number. These issues have been addressed by the Yearn Finance team and changes were made in the respective PRs.

While the access control mechanisms were generally well-designed, the review recommended imposing restrictions on the vault role opening feature to enhance security. Additionally, although the vault contract was well-documented, the tokenized-strategy suffered from a lack of NatSpec documentation on several functions.

The reviewers also highlighted the need to expand code coverage by integrating actual strategies in the testing and verification phase. Lastly, the reviewers stressed the need for more decentralization in the management of vault functionalities.

The review does not guarantee the security of the code and potential vulnerabilities may still exist. The Yearn Finance team and users of the contracts are advised to use the code at their own risk.
