

STATE MIND

Yearn V3

05-02-2024 – 01-03-2024

Table of contents



1. Project brief		2
2. Finding severity breakdown		3
3. Summary of findings		4
4. Conclusion		4
5. Findings report		5
Medium	DoS of report due to incorrect locking period calculation	5
	More strict loss checking	6
Informational	Unreachable/unused part of code	6
	Gas optimizations: unnecessary actions	7
	Code style: events	7
	setMaxProfitMaxUnlockTime can be sandwiched if called with 0	7
	Lack of zero address validation	8
	Proposal to create a more accurate previewRedeem function.	8
	WITHDRAW_LIMIT_MANAGER can stop all user withdrawals from strategies	9
	Gas optimization : code simplicity	9
	Lack of checking strategies for uniqueness	10
	Non-consistent variable naming in comments and code	11
	Vaults deposits DoS	12
	Possible PPS manipulations	14

1. Project brief



Title	Description
Client	Yearn
Project name	Yearn V3
Timeline	05-02-2024 - 01-03-2024

[tokenized-strategy](#).

Initial commit	b2ad48b34ef762778e4a49c8942a9910612c9cf9
Final commit	b2ad48b34ef762778e4a49c8942a9910612c9cf9

[yearn-vaults-v3](#)

Initial commit	f869f7e4ef18ff352294cf03200f7cb40df120b2
Final commit	d8abf371ef5570ec6f09336ed14526d022d1f5fb

Short Overview


Yearn Finance is DeFi’s premier yield aggregator. Giving individuals, DAOs and other protocols a way to deposit digital assets and receive yield.


Yearn Vaults (or yVaults) are capital pools that automatically generate yield based on opportunities present in the market. Vaults benefit users by socializing gas costs, automating the yield generation and rebalancing process, and automatically shifting capital as opportunities arise.


V3 sees the introduction of "Tokenized Strategies". Strategies are now capable of being standalone 4626 vaults themselves. These single-strategy vaults can be used as stand-alone vaults or easily added as a strategy to any of the multi-strategy "Allocator Vaults".


Project Scope

The audit covered the following files:

 [BaseStrategy.sol](#)

 [TokenizedStrategy.sol](#)

 [VaultFactory.vy](#)

 [VaultV3.vy](#)

2. Finding severity breakdown



All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds to be transferred to any party.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss of funds.
Informational	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Client regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Client is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

3. Summary of findings



Severity	# of Findings
Critical	0 (0 fixed, 0 acknowledged)
High	0 (0 fixed, 0 acknowledged)
Medium	2 (2 fixed, 0 acknowledged)
Informational	12 (6 fixed, 6 acknowledged)
Total	14 (8 fixed, 6 acknowledged)

4. Conclusion



During the audit of the codebase, 14 issues were found in total:

- 2 medium severity issues (2 fixed)
- 12 informational severity issues (6 fixed, 6 acknowledged)

The final reviewed commits are d8abf371ef5570ec6f09336ed14526d022d1f5fb ,
b2ad48b34ef762778e4a49c8942a9910612c9cf9

Deployment

Contract	Address
VaultV3	0x1ab62413e0cf2eBEb73da7D40C70E7202ae14467
Vault Factory	0x444045c5C13C246e117eD36437303cac8E250aB0
TokenizedStrategy	0xBB51273D6c746910C7C06fe718f30c936170feD0

5. Findings report



MEDIUM-01

DoS of report due to incorrect locking period calculation

Fixed at:
[d8abf37](#)

Description

In the calculation of locking period in **VaultV3._process_report** function, there is an error that can cause DoS of report processing. In case when **shares_to_lock > shares_to_burn** the amount of **shares_to_burn** is subtracted from **shares_to_lock** so it won't get considered in the calculation of the locking period (so fees or losses are not locked). However, in the case of **shares_to_lock <= shares_to_burn** there is an incorrect code behavior. Instead of setting **shares_to_lock** to zero, it sets **shares_to_burn** to zero.

```
# Adjust the amount to lock for this period.
if shares_to_lock > shares_to_burn:
    # Don't lock fees or losses.
    shares_to_lock = unsafe_sub(shares_to_lock, shares_to_burn)
else:
    shares_to_burn = 0 # <-- incorrect

...

previously_locked_time: uint256 = 0
_full_profit_unlock_date: uint256 = self.full_profit_unlock_date
if _full_profit_unlock_date > block.timestamp:
    previously_locked_time = (total_locked_shares - shares_to_lock) * (_full_profit_unlock_date - block.timestamp)
    # ^-- will revert in case when total_locked_shares < shares_to_lock

new_profit_locking_period: uint256 = (previously_locked_time + shares_to_lock * profit_max_unlock_time) /
total_locked_shares
# ^-- will improperly extend locking period in case when total_locked_shares >= shares_to_lock
```

This leads to the block of report processing under specific conditions, specifically when there are some shares still locked in the Vault and when **total_locked_shares < shares_to_lock < shares_to_burn**.

The ways to resolve this revert are:

- Wait till the full profit unlock date. This requires blocking protocol for reports of all strategies because otherwise, they will extend this period. During this period users can't earn any profit or distribute the losses (this may lead to withdrawal races).
- Manually set **self.full_profit_unlock_date** to zero via **setProfitMaxUnlockTime**. This way requires burning all locked shares, which is undesirable and may lead to possible sandwiches.

Recommendation

It is recommended to correct the condition behaviour:

```
# Adjust the amount to lock for this period.
if shares_to_lock > shares_to_burn:
    # Don't lock fees or losses.
    shares_to_lock = unsafe_sub(shares_to_lock, shares_to_burn)
else:
    - shares_to_burn = 0
    + shares_to_lock = 0
```

This adjustment ensures that the new profit locking period remains unchanged in case when we shouldn't extend it.

MEDIUM-02	More strict loss checking	Fixed at: d8abf37
<div><div>Description</div><p>Line: VaultV3.vy#L642 – this check is more or less strict than it actually should be.</p><p>If the purpose of max_loss is to set the current percentage of acceptable losses for each strategy separately, then we do not need to take into account the cumulative loss and have in this sum. If we need to look at the losses as a whole, then there is no point in doing this check in a loop and just comparing the total loss with max_assets. If you want to interrupt the cycle when the losses begin to exceed the limit in a particular strategy, then you need an additional coefficient that recalculates the share for withdrawal from max_assets and compares it with it.</p><p>Depending on this, there are several options for how to change this comparison.</p><p>Reasons why this comparison may be incorrect:</p><ol style="list-style-type: none">1. if the order of strategies is chosen poorly.2. if check here and actions inside will lead to a disproportionate change in variables.<p>This will result in incorrect calculation of maxRedeem() and maxWithdraw() functions.</p><div>Recommendation</div><p>We recommend replacing this sum with the max_assets variable.</p><div><div>– if loss + unrealised_loss > (have + to_withdraw) * max_loss / MAX_BPS:</div><div>+ if loss + unrealised_loss > max_assets * max_loss / MAX_BPS:</div><div># If so use the amounts up till now.</div><div>break</div></div></div>		

INFORMATIONAL-01	Unreachable/unused part of code	Fixed at: d8abf37
<div><div>Description</div><p>Lines:</p><p>VaultFactory.vy#L32 – unused interface</p><p>VaultV3#519 – unreachable condition. total_assets will always be greater than the amount, cause we sum self.total_idle with these assets earlier.</p><div>Recommendation</div><p>We recommend removing unused parts of code.</p></div>		

INFORMATIONAL-02	Gas optimizations: unnecessary actions	Fixed at: d8abf37
<p>Description</p> <p>Lines:</p> <p>VaultV3.vy#L304 – this part of code doesn't need to convert uint8 to uint256 and check that it's not greater than border, it's useless. In times of VV#-2020-0001 uint8 type didn't exist, but now it exists in vyper.</p> <p>VaultV3.vy#379 – this part of code doesn't need to convert v, r and s of signature to uint256, cause they already have a valid type.</p> <p>VaultV3.vy#L1413 – DEPOSIT_LIMIT_MANAGER role should set deposit_limit to max_value(uint256) to set the deposit limit module, but this will lead to the fact, that two functions should be called sequentially in one transaction, otherwise it may lead to problems. It's at least not optimized, as the maximum can be abused by malicious actors to deposit more than the current limit.</p> <p>Recommendation</p> <p>We recommend removing these convert functions and unnecessary check.</p>		

INFORMATIONAL-03	Code style: events	Fixed at: d8abf37
<p>Description</p> <p>Lines:</p> <p>VaultFactory.vy#L198</p> <p>VaultFactory.vy#L215</p> <p>Events should be logged after their formal change, and not before it.</p> <p>It will also be optional to write the current copy of the variable to memory, carry out all comparisons with it, and use it in logging; this will reduce the number of readings from the storage by 1.</p> <p>Recommendation</p> <p>We recommend reordering operations for better code style and caching self.default_protocol_fee_config for gas optimization.</p>		

INFORMATIONAL-04	setProfitMaxUnlockTime can be sandwiched if called with 0	Acknowledged
<p>Description</p> <p>It is possible to set self.profit_max_unlock_time to 0 in setProfitMaxUnlockTime. Vault's locked shares burn. This action increases price per share immediately.</p> <ol style="list-style-type: none">1. Attacker can find the setProfitMaxUnlockTime(0) transaction in the mem-pool and he could frontrun it to mint as much shares as he can.2. Then, attacker would redeem all the shares for assets at a higher price per share value. <p>This will cause a dilution of profits.</p> <p>Recommendation</p> <p>We recommend calling this function through private pool to prevent sandwiches.</p> <p>Client's comments</p> <div><p>Known and acknowledged.</p><p>The comments in the function specify this as a warning</p></div>		

INFORMATIONAL-05	Lack of zero address validation	Fixed at: d8abf37
------------------	---------------------------------	--------------------------------------

Description

The **initialize()** function doesn't check that the **role_manager** address does not equal zero. If the **role_manager** address is equal to zero, then it will not be possible to add users with other roles and it will not be possible to call **add_strategy()** and other privileged functions. That can lead to the redeployment of contract.

Vault's **mint()** and **deposit()** functions are called **_mint()** and **_deposit()** related functions. There is no address zero-check for the **receiver**. This can lead to the mint of shares to an empty address and a meaningless total_supply increase.

Recommendation

We recommend adding a zero-check for addresses in those functions.

INFORMATIONAL-06	Proposal to create a more accurate previewRedeem function.	Acknowledged
------------------	--	--------------

Description

Line: [VaultV3.vy#L2029](#)

The **previewRedeem()** function in **Vault** contract might show inaccurate preview information on the outcomes of redeem compared to actual **_redeem()** function. This happens, due to redeem process depending on the performance of the strategies in the queue. Therefore, the redeem amount out sometimes might be less than the amount expected by the user who depends on the **previewRedeem()** because of the unrealised losses of strategies. Thus, he might spend unnecessary gas cost on transactions that will be reverted due to unacceptable **max_loss** parameters.

The following scenario shows the discrepancy in **previewRedeem()** and actual **_redeem()**:

- Bob previews redeem: expects 30,000 assets.
- Actual redeem: receiving 27,500 assets.
- Difference: 2,500 assets (due to lossy strategy).
- Max-loss tolerance: 1,500 assets. (5% from 30,000)
- Result: **_redeem()** reverts due to exceeding **max_loss** tolerance.

Recommendation

We propose the creation of a function e.g., named **emulateRedeem()** which will emulate an actual redeem process based on the users desired arguments - **shares**, **strategies[]**, **max_loss**. This function would provide more accurate information on the **_redeem()**, thus saving the users from unnecessary gas costs.

Client's comments

Risk accepted.

We do not reccomend using preview functions except if needed for rounding preferences.

INFORMATIONAL-07	WITHDRAW_LIMIT_MANAGER can stop all user withdrawals from strategies	Acknowledged
------------------	--	--------------

Description

WITHDRAW_LIMIT_MANAGER can put limits on withdrawals from the strategies e.g., setting the maximum withdrawal amount in `withdraw_limit_module` to 0.

Impact:

The **WITHDRAW_LIMIT_MANAGER** role can block users from withdrawing funds by setting withdrawal limits to 0

Line: [VaultV3.vy#L586](#)

Recommendation

We recommend introducing minimum withdrawal thresholds for `withdraw_limit_module` agreed upon by community governance.

Client's comments

Known and acknowledged

INFORMATIONAL-08	Gas optimization : code simplicity	Fixed at: d8abf37
------------------	------------------------------------	-----------------------------------

Description

There is a check in `_redeem` for `withdraw_limit_module`, but in `_max_withdraw` we also have this check.

Recommendation

We recommend improving this part of the `_redeem` function and not calling `_max_withdraw`, but writing all logic in the if statement.

```
if self.withdraw_limit_module != empty(address):
+   max_assets: uint256 = self._convert_to_assets(self.balance_of[owner], Rounding.ROUND_DOWN)
+   limit: uint256 = min(
+       IWithdrawLimitModule(withdraw_limit_module).available_withdraw_limit(owner, max_loss, strategies),
+       max_assets
+   )

-   assert assets <= self._max_withdraw(owner, max_loss, strategies), "exceed withdraw limit"
+   assert assets <= limit, "exceed withdraw limit"
```

Description

The **Vault** contract contains a **set_default_queue()** function in which **QUEUE_MANAGER** can assign a new strategy queue. In addition to checking strategies for activation, it is necessary to check that the queue contains exclusively unique strategies. The absence of this check may lead to incorrect calculation of the **_max_withdraw()** function:

```
@view
@internal
def _max_withdraw(
    owner: address,
    max_loss: uint256,
    strategies: DynArray[address, MAX_QUEUE]
) -> uint256:

    // ...

    if max_assets > current_idle:

        // ...

        for strategy in _strategies:
            # Can't use an invalid strategy.
            assert self.strategies[strategy].activation != 0, "inactive strategy"

            # Get the maximum amount the vault would withdraw from the strategy.
            to_withdraw: uint256 = min(
                # What we still need for the full withdraw.
                max_assets - have,
                # The current debt the strategy has.
                self.strategies[strategy].current_debt
            # ^ -- current_debt is not being updated
            )

            // ...

        // ...

    return max_assets
```

Given that this is a preview method, it does not update the **current_debt** for each strategy. Therefore, if the strategy is repeated several times in the queue, **current_debt** will be counted several times. This leads to the fact that this method can withdraw a larger value than we can actually withdraw.

In addition, it is necessary to check the uniqueness of the strategies, if the user passes his unique queue.

Recommendation

We recommend adding a strategy uniqueness check to both the standard queue and the custom queue.

Client's comments

Risk accepted.

Vault managers should know not to reuse strategies and can remedy any mistake easily if need be.

The cost to verify each strategy is not worth the risk of someone getting a incorrect value if they are using it incorrectly.

Comments have been added to the `set_default_queue` and `maxRedeem/maxWithdraw` functions though to explain this risk.

INFORMATIONAL-10

Non-consistent variable naming in comments and code

Fixed at:
[d8abf37](#)

Description

In `_assess_share_of_unrealised_losses_comments` variable is called `assets_to_withdraw`, but in code `assets_needed` are used. It looks like it's the same entity called differently.

Recommendation

We recommend making code and comments consistent in this part of the code.

Description

In the **VaultV3**, there's a case when the total assets within the vault can become zero while the total supply of shares remains above zero. This case can happen by a report that decreases the total assets to zero after some deposits have been made. Under these conditions, both the **VaultV3.mint** and **VaultV3.deposit** functions will revert, thus making contract unavailable for deposits.

For **VaultV3.mint**:

```
@internal
def _convert_to_assets(shares: uint256, rounding: Rounding) -> uint256:
    if shares == max_value(uint256) or shares == 0:
        return shares

    total_supply: uint256 = self._total_supply()

    if total_supply == 0:
        return shares

    numerator: uint256 = shares * self._total_assets()
    amount: uint256 = numerator / total_supply
    if rounding == Rounding.ROUND_UP and numerator % total_supply != 0:
        amount += 1

    return amount # <-- will return zero

@internal
def _mint(sender: address, recipient: address, shares: uint256) -> uint256:
    assets: uint256 = self._convert_to_assets(shares, Rounding.ROUND_UP)
    assert assets > 0, "cannot deposit zero" # <-- will revert here
    ...
```

For **VaultV3.deposit**:

```

@internal
def _issue_shares_for_amount(amount: uint256, recipient: address) -> uint256:
    ...

    if total_supply == 0:
        new_shares = amount
    elif total_assets > amount:
        new_shares = amount * total_supply / (total_assets - amount)

    if new_shares == 0:
        return 0 # <-- return zero
    ...

@internal
def _deposit(sender: address, recipient: address, assets: uint256) -> uint256:
    ...
    shares: uint256 = self._issue_shares_for_amount(assets, recipient)
    assert shares > 0, "cannot mint zero" # <-- will revert here
    ...

```

In both cases, the calculations rely on the **total_assets** being greater than zero, leading to a deadlock scenario where no new deposits can be made if the total assets fall to zero. The only way to get out of DoS is to manually increase balance on some strategy and make **VaultV3.process_report** to increase total assets.

Recommendation

It is recommended to implement a mechanism that prevents the vault from entering a DoS state, such as:

- Implement a minimum threshold for total assets within **VaultV3.process_report**.
- Add a recovery function that allows vault administrators to manually increase **total_assets**.

Client's comments

Known.

This was done on purpose. This should only happen if a vault takes on a 100% loss and is desired to have the default state in that scenario be to consider the vault "dead" and no longer allow depositors.

However, it still is possible already for vault managers to increase total_assets giving a refund through the accountant back into the vault if desired to continue to use the vault

Description

The **VaultV3.process_report** function is designed to lock in profits by preventing the price per share (PPS) from increasing during its execution. This design aims to prevent transaction sandwiching, where an attacker buys shares at a lower price before **process_report** transaction and sells them at a higher price afterward, thereby diluting profits for other shareholders. However, this mechanism does not protect against reports of losses, which introduces a potential vulnerability when a strategy used by the Vault is susceptible to price manipulation.

An attacker could manipulate the price of some strategy's shares just before **VaultV3.process_report** is called. By manipulating the value of these shares, the Vault's PPS would decrease, enabling the attacker to purchase Vault shares at a reduced price.

Recommendation

It is recommended to implement a maximum allowable decrease limit (in percentages) for strategy losses to prevent large changes in a single block. If there is a large rebase in the strategy, it can be accounted for in several **process_report**. Also, consider using private mempools for **process_report** transactions.

Client's comments

Known.

Vault managers are responsible for assuring that the strategies PPS can not be manipulated and is why the V3 strategies are built in the same manner as the multi strategy vaults.

For further protection we (and would recommend others) implement "Health Checks" in both the strategy and vault reports that do these exact checks.

The current accountant in use in our vaults have a "max_gain" and "max_loss" variable that runs checks during process_report to make sure the report is within the expected range.

These same checks can be run at the strategy level to provide further assurances.

STATE MIND