Yo Protocol
Security Audit

Report Version 1.1

January 21, 2025

Conducted by **Hunter Security**

Table of Contents

1 About Hunter Security

Hunter Security is an industry-leading smart contract security company. Having conducted over 100 security audits protecting over $1B of TVL, our team delivers top-notch security services to the best DeFi protocols. For security audit inquiries, you can reach out on Telegram or Twitter at *@georgehntr*.

2 Disclaimer

Audits are a time-, resource-, and expertise-bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can reveal the presence of vulnerabilities, but cannot guarantee their absence.

3 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - involves a small loss of funds or affects a core functionality of the protocol.
- **Low** - encompasses any unexpected behavior that is non-critical.

3.2 Likelihood

- **High** - a direct attack vector; the cost is relatively low compared to the potential loss of funds.
- **Medium** - only a conditionally incentivized attack vector, with a moderate likelihood.
- **Low** - involves too many or unlikely assumptions; offers little to no incentive.

3.3 Actions required by severity level

- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

4 Executive summary

Overview

Project Name	Yo Protocol
Repository	https://github.com/yoprotocol/core
Commit hash	16d83ac2c602fc80e3e6a712b05aade4686228f5
Resolution	46d46e74ad99f905911460d5fd515a9f08c13252
Methods	Manual review & testing

Scope

src/yoVault.sol
src/AuthUpgradable.sol
src/Escrow.sol

Issues Found

High risk	0
Medium risk	0
Low risk	5
Informational	6

5 Findings

5.1 Low

5.1.1 Total pending assets not accounting for fees

Severity: Low

Files: src/yoVault.sol

Description: Consider the following scenario:

1. Let's say _getAvailableBalance is currently 105 ETH.
2. We *requestRedeem* for shares with a value of 110 ETH (including fees).
3. Let's say the fee is 10%.
4. 110 * 10 / 110 = 10 ETH fee, therefore 100 ETH net assets.
5. (105 > 110) == false, therefore we move to pending redeems.
6. *totalPendingAssets += 100*.
7. *_getAvailableBalance* is now equal to 5 ETH (105 - 100).

There were only 105 ETH available, we requested a redeem for 110 ETH which moved to pending, but 5 ETH remained as available. Now other users have the opportunity to withdraw up to 5 ETH even though they should be next on the queue.

Recommendation: Consider increasing *totalPendingAssets* in *requestRedeem* (and then decreasing in *cancelRedeem* and *fulfillRedeem*) by *assetsWithFee* instead of just *assets*.

Resolution: Resolved.

5.1.2 Incorrectly handling the case where fee recipient is not set

Severity: Low

Files: src/yoVault.sol

Description: The code comments state that no fee should be taken if the *feeRecipient* is *address(0)*, yet the smart contract does not implement this check.

Recommendation: Consider refactoring the fee logic to not take fee when no *feeRecipient* is set.

Resolution: Resolved.

5.1.3 Arbitrage opportunity if underlying balance update is delayed

Severity: Low

Files: src/yoVault.sol

Description: If there happens to be a time window/gap between funds leaving the yoVault's balance and the oracle updating the *aggregatedUnderlyingBalances*, an adversary may exploit the temporary incorrect value of *totalAssets()* so that they either withdraw more assets than usual or receive more shares upon deposit due to the flawed ratio.

Recommendation: Consider ensuring that the *aggregatedUnderlyingBalances* is always updated within the same transaction when funds are leaving the vault's balance.

Resolution: Acknowledged.

5.1.4 requestRedeem does not support approvals required by EIP4626

Severity: Low

Files: src/yoVault.sol

Description: EIP4626 states the following about *redeem* and *withdraw*: "MUST support a redeem flow where the shares are burned from owner directly where msg.sender has EIP-20 approval over the shares of owner". However, this flow is not implemented in the yoVault.

Recommendation: Consider allowing users to request redeeming shares for others if approval has been granted in order to comply with the EIP requirements.

Resolution: Acknowledged.

5.1.5 Users not able to cancel their withdrawal requests by themselves

Severity: Low

Files: src/yoVault.sol

Description: Users should be able to cancel their own redeems if a long time has passed and their request has not been fulfilled yet as the price-per-share may have significantly changed so that they no longer intend to redeem their shares at the initially requested ratio. They might as well prefer to sell or exchange them on a secondary market if there is a delay in fulfilling their request.

Recommendation: Consider either allowing users to cancel their own redeem requests or introducing a deadline parameter or refactoring the logic so that the assets:shares ratio is calculated upon fulfilling the request rather than upon creation.

Resolution: Acknowledged.

5.2 Informational

5.2.1 Typographical mistakes, non-critical issues or centralization vulnerabilities

Severity: Informational

Files: src/yoVault.sol

Description: The contracts contain one or more typographical mistakes, non-critical issues or centralization vulnerabilities. In an effort to keep the report size reasonable, we enumerate these below:

1. Users shouldn't be able to pass *address(0)* as a *receiver* in *requestRedeem*.
2. The *msg.sender* is passed to *_withdraw* in *requestRedeem* instead of the *receiver*.
3. Misleading comment in *cancelRedeem* stating that shares are returned to the owner while they are sent to the receiver.
4. *updateWithdrawFee*, *updateDepositFee*, and *updateMaxPercentageChange* should use <=.
5. *maxDeposit* and *maxMint* must return 0 when the protocol is paused as per EIP4626.
6. The concept of having a request Id appears to be unnecessary as it always has a value of 0.

Recommendation: Consider fixing the above typographical mistakes, non-critical issues or centralization vulnerabilities.

Resolution: Partially resolved (only pt. 2).