Rari Vaults Review

Dhurv, Kat, Amanusk

Frontrunning loss

Withdrawals above vaults loose assets starts withdrawing from individual strategies, there does not seem to be any loss handling logic we would expect to see some

form of invariant of the kind: *amountNeeded - loss <= amountWithdrawn*. Missing this sort of protection means that any kind of loss can be frontrun

example:

If a strategy sufferers a loss, and harvest is not called before someone make a withdraw, there will be a race to withdraw the funds first.
New users depositing funds into the vault, could have their funds removed by existing depositors, and leave them empty handed

```
function testLossOnStrategyAndWithdraw() public {
    underlying.mint(address(user1), 1e18);
    underlying.mint(address(user2), 1e18);

    user1.approve(address(vault), 1e18);
    user2.approve(address(vault), 1e18);

    uint256 preDepositBal = underlying.balanceOf(address(user1));

    user1.deposit(1e18);

    vault.trustStrategy(strategy1);
    vault.depositIntoStrategy(strategy1, 1e18);
    vault.pushToWithdrawalQueue(strategy1);

    strategy1.simulateLoss(1e18);

    user2.deposit(1e18);

    user1.withdraw(1e18);
    assertEq(underlying.balanceOf(address(user1)), 1e18);
}
```

User 1 can withdraw the funds, although all the funds were lost.
User 2 will not be able to withdraw anything after their deposit

Strategies with funds popped out of queue

Removing strategies from the withdraw queue happens upon withdraw. The removed amount depends on the last reported balance of the strategy.
If the strategy has gained funds since the last harvest (e.g. rebasing tokens/airdrop), it will be removed from the queue, but still have funds in it.

This might lead to inaccessible fund of strategies in the queue (or require manual reinsertion).

Also not sure why pullFromWithdrawalQueue removes strategies from the withdrawal queue - it feels like this should be a separate concern - check if a strategy is empty, when withdrawing, but if it needs to be removed, remove it through a separate function

```
function testHoleInStrategyRemoval() public {
    underlying.mint(address(user1), 2.5e18);
    underlying.mint(address(user2), 1e18);

    user1.approve(address(vault), 2.5e18);
    user2.approve(address(vault), 1e18);

    uint256 preDepositBal = underlying.balanceOf(address(user1));

    user1.deposit(2.5e18);
    user2.deposit(0.1e18);

    assertEq(vault.totalHoldings(), 2.6e18);

    vault.trustStrategy(strategy1);
    vault.depositIntoStrategy(strategy1, 1e18);
    vault.pushToWithdrawalQueue(strategy1);

    vault.trustStrategy(strategy2);
    vault.depositIntoStrategy(strategy2, 0.6e18);
    vault.pushToWithdrawalQueue(strategy2);

    vault.trustStrategy(strategy3);
    vault.depositIntoStrategy(strategy3, 1e18);
    vault.pushToWithdrawalQueue(strategy3);

    underlying.mint(address(user2), 1e18);
    user2.transfer(address(strategy2), 0.1e18);

    user1.withdraw(2.49e18);

    vault.harvest(strategy2);

    user1.withdraw(0.02e18);
}
```

User 1 will be have access to either >2.5 or <2.5 depending on when harvest is called and strategy balance is updated

Fuzzing

Deposits:
 - underlyingAmount.fdiv(exchangeRate(), BASE_UNIT)) depends on exchange rate, exchange rate can fail if amount of rc tokens > 0, but underlyingAmount == 0;

Withdrawals:
 - ExchangeRate in strategies, if one strategy suffers a complete loss of assets the exchangeRate calculation will revert, blocking any withdrawals until this strategy has
 been removed from the withdrawalqueue.

General design / feedbacks (by Dhruv)

1. Nature of constant parameters function like ProfitUnlockDelay
    a. Whether should they be entrusted to only msg.sender , as if in case its sets it high value and then “strategy” becomes defunct -> might be an issue of central loss.
    b. Can there be possibility of using timelock instead ?
2. Potential issues regarding the malicious issues being addressed in comments .
    a. Like Checking the nature of token in strategy in the strategy .

3. Well written / sourcify standard natspec