Review of Rari Vaults

Nibbler, bebis, Zokunei, Carl Farterson

General stuff

Is there a reason vault shares were minted before funds were transferred from the user? Just general bad practice, especially with no reentrancyGuard. Don’t assume you will always control context.

Added complexity with custom libraries - non-standard code/patterns affect readability, which is more important for etdev than any other programming paradigm. I think you should reassess tradeoffs just to ensure the benefits of modularity outweigh the benefits of users easily understanding operations

Line 274, use BASE_UNIT >:^)

Context issues

Use of msg.sender throughout - vault should better control context in general msgSender()

No way to account for tokens sent to the strategy by means not pre-determined by vault logic. Whether through air drops or mistakes, capturing value that enters the vault ecosystem unpredictably is quality UX

In lockedProfit, profitUnlockDelay can be changed before all existing profits are unlocked, either put a cap on profitUnlockDelay or require block.timestamp>=lastHarvest+profitUnlockDelay before being able to update

Withdraw/Redeem

Withdraw/redeem do not call harvest or otherwise check whether there were losses or gains before allowing people to withdraw/redeem. At best, users have to remember to call harvest before they withdraw/redeem. At worst, this allows users to skip out on any losses that may have occurred before they withdraw/redeem.

Harvest / trust system / auth

At best connascence of name w/ balanceOfUnderlying on both vault and strategy - at worst the harvest function is fairly useless or a good way to mess up the accounting of the entire system. Should there be a strategy.harvest() in between lines 273 and 274? Seems like optimized yearn v1 so that’s where we’re coming from, would like an explanation of design decisions/tradeoff analysis

Authentication and trust/mistrust system seems a little useless, we are missing a lot of context, couldn’t the admin both trust a strategy and add an evil strategy to the queue?

Line 377 seems to fix the state issue w/r/t updating the balance of strategy

If harvest is spammed, the vesting time can be extended beyond the expected 6 hours, as lastHarvest is continuously updated - should maybe requireAuth

deposit/withdraw to/from strategy

Should maybe not allow invalid deposits/withdraws - validate inputs

There should be a way to validate the totalStrategyHoldings before updating state - a function that recalcs every time by calling dependencies and whatnot and determining how many tokens are TRULY in the system

WTF is mint???? There is approval but no transfer

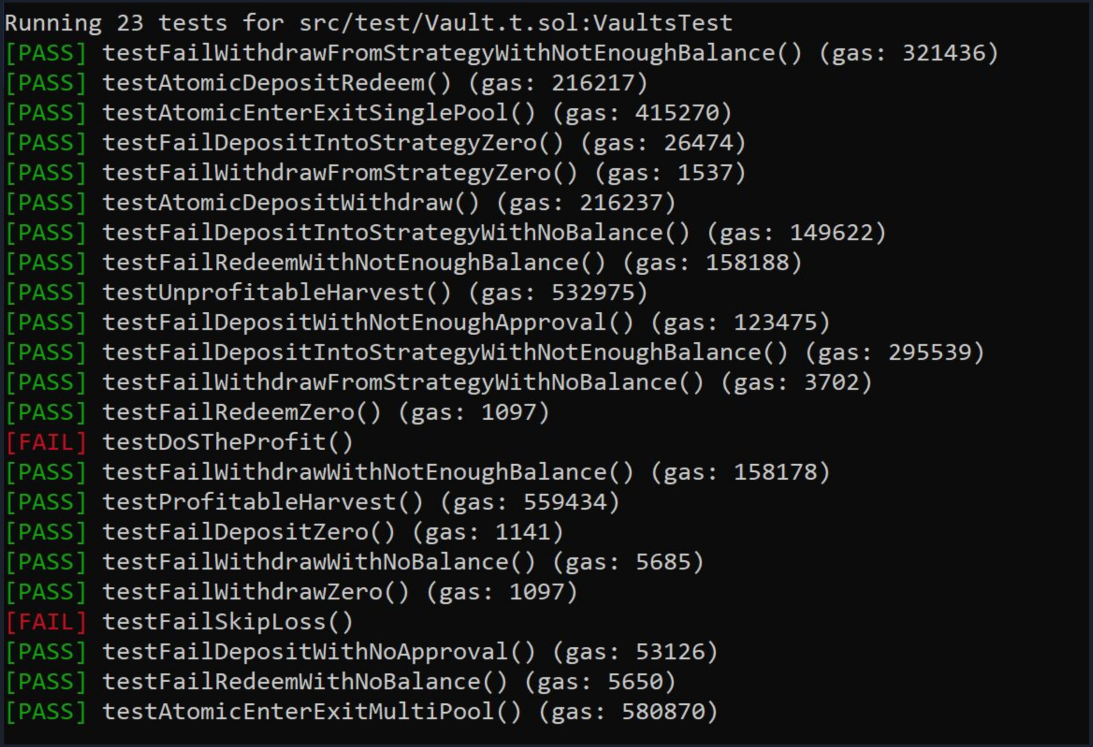
There isn't any pushing or popping of strategies from the queue, so a lot of dependent processes are disjointed

Same issues for withdraw

Conclusion

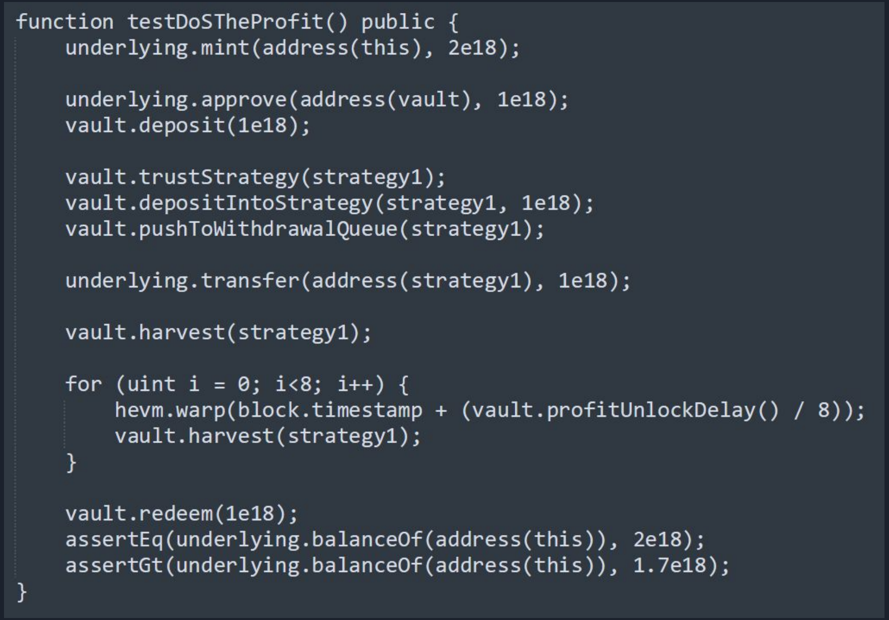
It seems like the designer favored modularity and encapsulation more than security. A lot of the accounting and state changes need to be handled manually by a trusted centralized party, which removes a lot of the decentralized, anti-fragile elements of the original Yearn V1 system. We believe tradeoffs should be better analyzed with security and reliability in mind as opposed to granularity and modularity. It’s definitely respectable how organized the system is, but there are a lot of benefits to monolithic architectures that are lost along the way.

Dapp Tools Bonus - Test Results

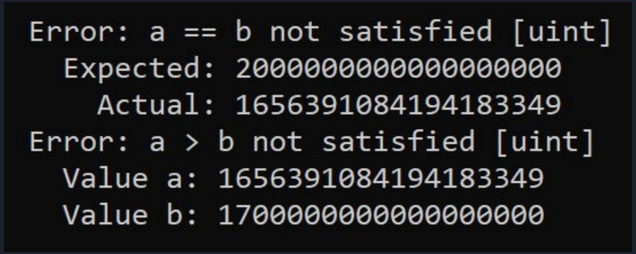


```
Running 23 tests for src/test/Vault.t.sol:VaultsTest
[PASS] testFailWithdrawFromStrategyWithNotEnoughBalance() (gas: 321436)
[PASS] testAtomicDepositRedeem() (gas: 216217)
[PASS] testAtomicEnterExitSinglePool() (gas: 415270)
[PASS] testFailDepositIntoStrategyZero() (gas: 26474)
[PASS] testFailWithdrawFromStrategyZero() (gas: 1537)
[PASS] testAtomicDepositWithdraw() (gas: 216237)
[PASS] testFailDepositIntoStrategyWithNoBalance() (gas: 149622)
[PASS] testFailRedeemWithNotEnoughBalance() (gas: 158188)
[PASS] testUnprofitableHarvest() (gas: 532975)
[PASS] testFailDepositWithNotEnoughApproval() (gas: 123475)
[PASS] testFailDepositIntoStrategyWithNotEnoughBalance() (gas: 295539)
[PASS] testFailWithdrawFromStrategyWithNoBalance() (gas: 3702)
[PASS] testFailRedeemZero() (gas: 1097)
[FAIL] testDoSTheProfit()
[PASS] testFailWithdrawWithNotEnoughBalance() (gas: 158178)
[PASS] testProfitableHarvest() (gas: 559434)
[PASS] testFailDepositZero() (gas: 1141)
[PASS] testFailWithdrawWithNoBalance() (gas: 5685)
[PASS] testFailWithdrawZero() (gas: 1097)
[FAIL] testFailSkipLoss()
[PASS] testFailDepositWithNoApproval() (gas: 53126)
[PASS] testFailRedeemWithNoBalance() (gas: 5650)
[PASS] testAtomicEnterExitMultiPool() (gas: 580870)
```

Dapp Tools Bonus - DoSTheProfit



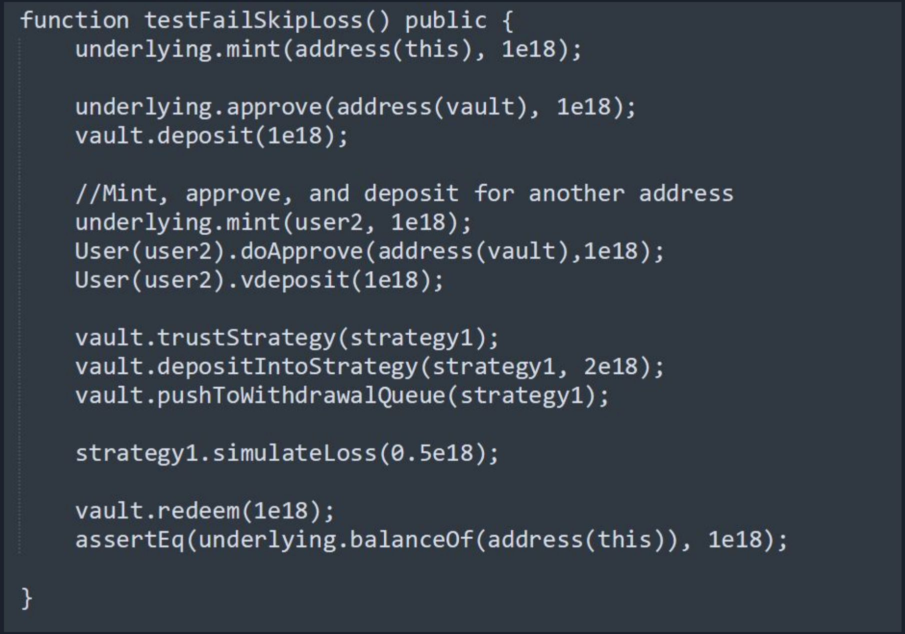
```
function testDoSTheProfit() public {
    underlying.mint(address(this), 2e18);

    underlying.approve(address(vault), 1e18);
    vault.deposit(1e18);

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

    underlying.transfer(address(strategy1), 1e18);

    vault.harvest(strategy1);

    for (uint i = 0; i<8; i++) {
        hevm.warp(block.timestamp + (vault.profitUnlockDelay() / 8));
        vault.harvest(strategy1);
    }

    vault.redeem(1e18);
    assertEq(underlying.balanceOf(address(this)), 2e18);
    assertGt(underlying.balanceOf(address(this)), 1.7e18);
}
```



```
Error: a == b not satisfied [uint]
  Expected: 2000000000000000000
    Actual: 1656391084194183349
Error: a > b not satisfied [uint]
  Value a: 1656391084194183349
  Value b: 1700000000000000000
```

Over 30% of the profit is still inaccessible at the end of the unlock delay.

Dapp Tools Bonus - SkipLoss



```
function testFailSkipLoss() public {
    underlying.mint(address(this), 1e18);

    underlying.approve(address(vault), 1e18);
    vault.deposit(1e18);

    //Mint, approve, and deposit for another address
    underlying.mint(user2, 1e18);
    User(user2).doApprove(address(vault),1e18);
    User(user2).vdeposit(1e18);

    vault.trustStrategy(strategy1);
    vault.depositIntoStrategy(strategy1, 2e18);
    vault.pushToWithdrawalQueue(strategy1);

    strategy1.simulateLoss(0.5e18);

    vault.redeem(1e18);
    assertEq(underlying.balanceOf(address(this)), 1e18);

}
```

Two users both put in 1e18. A loss of 0.5e18 occurs. One user sticks the other with the entire loss.