StrategyImperamaxLender Audit by Team2 (takeda)

General

Theorically we could skip all transfer/burn/mint require because if we transfer (want/btoken) nothing (0) the mint/redeem will anyway revert. We could add that just for the sake of avoiding external call and save gas.

TAR-XXX: `_reorderPools` is not updtating `preventDeposits`

Tools/Techniques: Manual
Difficulty+Impact: Critical Risk

Details

The current reorder functionality is updating the order of both `utilizations` and `newPoolSorting` but not `preventDeposits`. This would mean that after the reorder it would be possible that a pool that before was enabled to deposit into would be disabled and that a pool that was disabled would become enabled.

This would influence both the `deposit` and `attemptToRemovePool` that uses `preventDeposits` values.

Mitigation

In `_reorderPools` add also the reorder of the `preventDeposits` array like this `(preventDeposits[low1], preventDeposits[high1]) = (preventDeposits[high1], preventDeposits[low1]);`

TAR-XXX: `_deposit` are double decreasing the `i` index

Tools/Techniques: Manual
Difficulty+Impact: Critical Risk

Details

The `_deposit` function want to start depositing to the higher utilization rate pool (if the pool array have been correctly reordered) so it needs to start depositing to the pool in the higher index position of the array.

With the current implementation the loop index `i` is both decreased at the end of each loop but also at the start of the loop.

Let's make an example, pools len = 10;

```
preventDeposits[9] = true;
preventDeposits[8] = false;
preventDeposits[7] = false;
```

Before for: `i = pools.length` -> 10
Iteration 1: `i = pools.length-1` -> 9 because of `i = i.sub(1)`
Iteration 1: not enter if because the pool has `preventDeposits=true`
Iteration 1: end iteration -> `i = i--` -> i = 8
Iteration 2: because of `i = i.sub(1)` -> i = 7 and it would have skipped the pool on position `8` that would have been ok to deposit into because `preventDeposits[8] = false`

Mitigation

Change the for like this:

```
for (uint256 i = pools.length-1; i >= 0; i--) {
    if (!preventDeposits[i]) {
        // only deposit to this pool if it's not shutting down.
        address targetPool = pools[i];

        want.transfer(targetPool, _depositAmount);

        require(IBorrowable(targetPool).mint(address(this)) >= 0);
        break;
    }
}
```

TAR-XXX: `_initializeStrat` allows pools to drain `want` balance

Tools/Techniques: Manual
Difficulty+Impact: High Risk

Details

The `_initializeStrat` method approve the `tarot` pool to be able to withdraw the max amount of `want` token. This should be not allowed given that `tarot` pools don't need to manage the strategy/vault funds.

Mitigation

Remove `want.safeApprove(_pools[i], type(uint256).max);` from `_initializeStrat` function.

TAR-XXX: `_initializeStrat` is not validating `_pools` user input

Tools/Techniques: Manual
Difficulty+Impact: High Risk

Details

`_initializeStrat` should validate `_pools` user input in the same way that the `addTarotPool` is doing.

Inside `addTarotPool` the function is validating:

- the `pool` `underlying` token is the same one managed by the strategy (`want`)
- is checking that the pool that is being added to the array has not been already added to the array

Mitigation

Implements those checks also in the `_initializeStrat` function.

TAR-XXX: `trueExchangeRate` and all functions that use it could revert because of division by zero

Tools/Techniques: Manual
Difficulty+Impact: Informational

Details

If we look at Tarot `exchangeRate` implementation

```
function exchangeRate() public returns (uint256) {
    uint256 _totalSupply = totalSupply; // gas savings
    uint256 _totalBalance = totalBalance; // gas savings
    if (_totalSupply == 0 || _totalBalance == 0) return
initialExchangeRate;
    return _totalBalance.mul(1e18).div(_totalSupply);
}
```

we can see that they handle the case were the supply or the balance is zero to avoid inconsistent results and revert by division by zero.

The same checks should be made also in `trueExchangeRate` or at least manage the possible scenarios in all the functions that are using directly or inderectly `trueExchangeRate`

Mitigation

Correctly handle cases where `actualBalance` or `totalSupply` is zero in `trueExchangeRate` function.

TAR-XXX: General gas optimizations

Tools/Techniques: Manual
Difficulty+Impact: Informational/Gas Optimization

Details

1. in `prepareReturn` use the already calculated `debtNeeded` in the `_withdraw`

```
if (stakedBal > 0) {
    // don't bother withdrawing if we don't have staked funds
    uint256 debtNeeded = Math.min(stakedBal, _debtOutstanding);
    // @audit - use debtNeeded instead of recalc it
    _withdraw(debtNeeded);
}
```

2. in `prepareMigration` avoid `bToken.transfer` if `balanceOfbToken != 0`
3. `IBorrowable(pool).mint()` return value check if it's only like this (`>=0`) because internally `tarot` would already revert if `mintTokens == 0` (see Tarot.mint function)

Mitigation

Follow the suggested gas optimization

TAR-XXX: `manuallySetAllocations` should ensure to correctly set ratios based on pools utilizations

Tools/Techniques: Manual
Difficulty+Impact: Informational

Details

There are two solutions here:

1. offchain the pool utilizations have been already calculated and ratios passed as inputs already have taken in count of that
2. the ratios are already ordered ASC and the contract should also call `_reorderPools` before distributing the `want` token with the new ratios

Mitigation

See Details

TAR-XXX: `harvestTrigger` does not use `callCostInWei`

Tools/Techniques: Manual
Difficulty+Impact: Minor

Details

callCostInWei passed as an argument but used for anything

Mitigation

Remove it or use it, ignore this if it is part of the necessary keep3r api

TAR-XXX: `attemptToRemovePool` may not use the most accurate exchange rate

Tools/Techniques: Manual
Difficulty+Impact: Minor

Details

This function uses IBorrowable().exchangeRateLast(), this may lead to inconsistencies when withdrawing from the pools as the rate might have changed

Mitigation

Call updateExchange on the pool to update it

TAR-XXX: transfer calls can be changed to safeTransfer

Tools/Techniques: Manual
Difficulty+Impact: Misc

Details

The strategy uses `SafeERC20` for `IERC20` but uses the base `transfer` function instead of `safeTransfer`.
If there is enough confidence and knowledge of the ERC20 tokens the strategy interacts with, this may be redundant.

Mitigation

Update to the `SafeERC20` version of those functions