

Strategy Contract Vulnerability Report

AUM Fees Lost During Operator Transfer

MEDIUM SEVERITY

Summary

When there is transfer of operator ownership and the previous owner did not rebalance, the amassed AUM fee is collected by the new operator, the old operation gets nothing.

Finding Description

AUM fees are collected during rebalance operations through the `_withdrawAndApplyAumAnnualFee` function in the Strategy contract. The calculation of the fee is determined by the amount of time since the last rebalance.

When there is an operator change by calling the `setOperator` function, there is no change in the rebalance time if the pervious operator has not performed a rebalance during their time of control, the accumulated fee will be collected by the new operator during their first rebalance.

```
// Apply the AUM annual fee
if (lastRebalance < block.timestamp) {
    uint256 annualFee = _aumAnnualFee;

    if (annualFee > 0) {
        // pay the aum fee to the market maker if any, otherwise to the default
        address feeRecipient = _factory.getFeeRecipientByVault(_vault());

        // Get the duration of the last rebalance and cap it to 1 day.
        uint256 duration = block.timestamp - lastRebalance;
        duration = duration > 1 days ? 1 days : duration;

        // Round up the fees and transfer them to the fee recipient.
```

```
uint256 feeX = (totalBalanceX * annualFee * duration + _SCALED_YEAR_SUB.  
uint256 feeY = (totalBalanceY * annualFee * duration + _SCALED_YEAR_SUB.  
  
    // [Transfer code...]  
}  
}
```

This function checks who the current fee recipient is at the time of calling (`_factory.getFeeRecipientByVault(_vault())`), not who was the fee recipient during the period when fees accrued.

Impact Explanation

- Pervious operator losses the AUM fee durning the time as owner

Proof of Concept

A coded function in the StrategyTest contract:

```
function test_OperatorChangeFeeCollection() external {  
    // Create vault and strategy  
    depositToVault(vault, alice, 100e18, 100e6);  
  
    // Set the AUM fee to a significant value to make the test clearer  
    vm.prank(address(factory));  
    IStrategy(strategy).setPendingAumAnnualFee(0.2e4);  
  
    // Set initial operator  
    address initialOperator = makeAddr("initialOperator");  
    vm.prank(owner);  
    factory.setOperator(IStrategy(strategy), initialOperator);  
  
    // First rebalance to activate the fee  
    vm.prank(initialOperator);  
    IStrategy(strategy).rebalance(0, 0, 0, 0, 0, 0, new bytes(0));  
  
    // Advance time by 20 hours to accrue fees  
    vm.warp(block.timestamp + 20 hours);
```

```
// Change operator without rebalancing first
address newOperator = makeAddr("newOperator");
vm.prank(owner);
factory.setOperator(IStrategy(strategy), newOperator);

// Check balances before rebalance
address feeRecipient = factory.getFeeRecipientByVault(vault);
assertEq(feeRecipient, newOperator, "Fee recipient should be the new operator");
uint256 newOperatorBalanceXBefore = IERC20(wavax).balanceOf(newOperator);
uint256 newOperatorBalanceYBefore = IERC20(usdc).balanceOf(newOperator);
uint256 initialOperatorBalanceXBefore = IERC20(wavax).balanceOf(initialOperator);
uint256 initialOperatorBalanceYBefore = IERC20(usdc).balanceOf(initialOperator);

// New operator performs rebalance and collects fees
vm.prank(newOperator);
IStrategy(strategy).rebalance(0, 0, 0, 0, 0, 0, new bytes(0));


// Check that fees went to new operator, not initial operator
uint256 newOperatorBalanceXAfter = IERC20(wavax).balanceOf(newOperator);
uint256 newOperatorBalanceYAfter = IERC20(usdc).balanceOf(newOperator);
uint256 initialOperatorBalanceXAfter = IERC20(wavax).balanceOf(initialOperator);
uint256 initialOperatorBalanceYAfter = IERC20(usdc).balanceOf(initialOperator);

// Verify new operator received fees
assertGt(newOperatorBalanceXAfter, newOperatorBalanceXBefore, "New operator received token X fees");
assertGt(newOperatorBalanceYAfter, newOperatorBalanceYBefore, "New operator received token Y fees");

// Verify initial operator received nothing
assertEq(
    initialOperatorBalanceXAfter,
    initialOperatorBalanceXBefore,
    "Initial operator should not receive token X fees"
);
assertEq(
    initialOperatorBalanceYAfter,
    initialOperatorBalanceYBefore,
    "Initial operator should not receive token Y fees"
);
}
```

Recommendation

Modify the `setOperator` function to automatically perform a rebalance before



changing the operator.