HAECHI AUDIT

One Ring

Smart Contract Security Analysis

Published on : Mar 02, 2022

Version v2.0

HAECHI AUDIT

Smart Contract Audit Certificate

One Ring

Security Report Published by HAECHI AUDIT
v1.0 Jan 27, 2022
v2.0 Mar 02, 2022

Auditor : Felix Kim

Executive Summary

Severity of Issues	Findings	Resolved	Unresolved	Acknowledged	Comment
Critical	2	1	1	-	-
Major	4	2	2	-	-
Minor	1	-	1	-	-
Tips	-	-	-	-	-

TABLE OF CONTENTS

7 Issues (2 Critical, 4 Major, 1 Minor) Found

ABOUT US

HAECHI AUDIT believes in the power of cryptocurrency and the next paradigm it will bring.

We have the vision to *empower the next generation of finance*. By providing security and trust in the blockchain industry, we dream of a world where everyone has easy access to blockchain technology..

HAECHI AUDIT is a flagship service of HAECHI LABS, the leader of the global blockchain industry. HAECHI AUDIT provides specialized and professional smart contract security auditing and development services.

We are a team of experts with years of experience in the blockchain field and have been trusted by 300+ project groups. Our notable partners include Universe,1inch, Klaytn, Badger, etc.

HAECHI AUDIT is the only blockchain technology company selected for the Samsung Electronics Startup Incubation Program in recognition of our expertise. We have also received technology grants from the Ethereum Foundation and Ethereum Community Fund.

Inquiries: audit@haechi.io

Website: audit.haechi.io

INTRODUCTION

This report was prepared to audit the security of the smart contract created by One Ring team. HAECHI AUDIT conducted the audit focusing on whether the smart contract created by One Ring team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the smart contract.

CRITICAL	Critical issues must be resolved as critical flaws that can harm a wide range of users.
MAJOR	Major issues require correction because they either have security problems or are implemented not as intended.
MINOR	Minor issues can potentially cause problems and therefore require correction.
TIPS	Tips issues can improve the code usability or efficiency when corrected.

HAECHI AUDIT recommends One Ring team improve all issues discovered.

The following issue explanation uses the format of {file name}#{line number}, {contract name}#{function/variable name} to specify the code. For instance, *Sample.sol:20* points to the 20th line of Sample.sol file, and *Sample#fallback()* means the fallback() function of the Sample contract.

Please refer to the Appendix to check all results of the tests conducted for this report.

SUMMARY

The codes used in this Audit can be found at GitHub (https://github.com/OneRingUSD/OneRing-Contracts/tree/30a7677e1eee63e26cde31bbcaf7d99d66097a2f/contracts). The last commit of the code used for this Audit is "30a7677e1eee63e26cde31bbcaf7d99d66097a2f".

Issues	HAECHI AUDIT found 2 critical issues, 4 major issues, and 1 minor issue. There are 0 Tips issues explained that would improve the code's usability or efficiency upon modification.
Update	[v.2.0] Regarding new commit 4afe3517fe564ab9c3de9e410b5ddd26d5a96597 fixed 1 critical issue and 2 major issues.

Severity	Issue	Status
CRITICAL	The minting logic of the vault token fails to work properly.	(Found - v1.0)
CRITICAL	The MultiStrategy#withdrawAllToVault() function fails to work properly.	(Found - v1.0) (Resolved - v.2.0)
MAJOR	When the length of the underlyings array of the vault contract is not 3, the function may work improperly.	(Found - v1.0) (Resolved - v.2.0)
MAJOR	When changing the active strategy of OneRingVault contract, proper migration from the old strategy to the new strategy may fail to occur.	(Found - v1.0) (Resolved - v.2.0)

⚠ **MAJOR**	When there is a disabled strategy during the operation of OneRingVault utilizing MultiStrategy, the amount deposited per strategy may be inaccurate.	(Found - v1.0)
⚠ **MAJOR**	The withdrawToVault function withdraws abnormal values to the vault.	(Found - v1.0)
● **MINOR**	No syntax approves the asset to the router in the OneRingVault#addUnderlying() function.	(Found - v1.0)

OVERVIEW

Contracts subject to audit

- OneRingVault
- MultiStrategy
- MasterChefBaseStrategy
- MiniChefStrategy
- StakingRewardsBaseStrategy
- QuickPolygonUsdtUsdcStrategy
- SpookyFantomMaiUsdcStrategy
- SpookyFantomTusdUsdcStrategy
- SushiPolygonFraxUsdcStrategy
- SushiPolygonTusdUsdcStrategy

FINDINGS

CRITICAL

The minting logic of the vault token fails to work properly. (Found - v.1.0)

Issue

Through the vault contract, users deposit stable coins in the Vault and mint the Vault tokens according to the ratio between the Vault's USD balance and totalSupply.
At this time, due to an error in the ratio calculation logic, the minting of the Vault token may not be performed properly. The following is the test scenario.

Alice's usdt balance before deposit: **21146.578344**
Alice's Vault balance before deposit: **0**
Alice's usdt balance after first 2000$ deposit: **19146.578344**
Alice's Vault balance after first 2000$ deposit: **988212080766613**
Alice's usdt balance after second 2000$ deposit: **17146.578344**
Alice's Vault balance after second 2000$ deposit: **988212572358047**

In the scenario, when Alice first deposits 2000 usdt, she gets the vault balance of **988,212,080,766,613**; however, after depositing 2000 usdt as the second in the order, Alice's vault balance increases to **988,212,572,358,047**, which is a very small amount.

Recommendation

We advise modifying the ratio calculation logic.

CRITICAL

The MultiStrategy#withdrawAllToVault() function fails to work properly. (Found - v.1.0) (Resolved - v.2.0)

```
function withdrawAllToVault(address _asset) public restricted {
    for (uint256 _sid = 0; _sid < strategyInfo.length; _sid++) {
        IStrategy(strategyInfo[_sid].strategy).withdrawAllToVault();
    }
}
```

[https://github.com/OneRingUSD/OneRing-Contracts/blob/30a7677e1eee63e26cde31bbcaf7d99d66097a2f/contracts/strategies/MultiStrategy.sol#L137-L141]

Issue

The parameter is omitted when calling the *IStrategy(strategyInfo[_sid].strategy).withdrawAllToVault()* function from the *MultiStrategy#withdrawAllToVault()* function. Thus, this function cannot be used.

Recommendation

We recommend modification to *IStrategy(strategyInfo[_sid].strategy).withdrawAllToVault(_asset)*.

Update

[v2.0] - OneRing team resolved the issue by adding the parameter.

⚠ **MAJOR**

When the length of the underlyings array of the vault contract is not 3, the function may work improperly. (Found - v.1.0) (Resolved - v.2.0)

```
function initializeMasterChefBaseStrategy(
    address _vault,
    address _parentStrategy,
    address _underlying,
    address _masterChef,
    address _router,
    address _rewardToken,
    uint256 _poolId,
    uint256 _sellFloor
) public initializer {
    __Ownable_init();
    vault = _vault;
    parentStrategy = _parentStrategy;

    underlying = _underlying;
    masterChef = _masterChef;
    router = _router;
    rewardToken = _rewardToken;
    poolId = _poolId;

    sellFloor = _sellFloor;

    address _lpt;
    (_lpt, , , ) = IMasterChef(masterChef).poolInfo(poolId);
    require(_lpt == underlying, "Pool Info doesn't match underlying");

    IERC20(underlying).safeApprove(masterChef, 0);
    IERC20(underlying).safeApprove(masterChef, uint256(-1));

    address _token0 = IUniswapV2Pair(underlying).token0();
    address _token1 = IUniswapV2Pair(underlying).token1();

    IERC20(_token0).safeApprove(router, 0);
    IERC20(_token0).safeApprove(router, uint256(-1));

    IERC20(_token1).safeApprove(router, 0);
    IERC20(_token1).safeApprove(router, uint256(-1));
```

```
        IERC20(underlying).safeApprove(router, 0);
        IERC20(underlying).safeApprove(router, uint256(-1));

        IERC20(rewardToken).safeApprove(router, 0);
        IERC20(rewardToken).safeApprove(router, uint256(-1));

        for (uint256 i = 0; i < 3; i++) {
            address _asset = IVault(vault).underlyings(i);
            IERC20(_asset).safeApprove(router, 0);
            IERC20(_asset).safeApprove(router, uint256(-1));
        }
    }
```

[https://github.com/OneRingUSD/OneRing-Contracts/blob/30a7677e1eee63e26cde31bbcaf7d99d66097a2f/contracts/strategies/base/MasterChefBaseStrategy.sol#L37-L86]

```
    function _investAllAssets() internal {
        // address[] memory _vaultAssets = IVault(vault).underlyings();
        address _token0 = IUniswapV2Pair(underlying).token0();
        address _token1 = IUniswapV2Pair(underlying).token1();

        for (uint256 i = 0; i < 3; i++) {
            address _asset = IVault(vault).underlyings(i);
            address[] memory _route0 = fromVaultAssetRoutes[_asset][_token0];
            address[] memory _route1 = fromVaultAssetRoutes[_asset][_token1];
            _assetToUnderlying(_asset, _route0, _route1);
        }
    }
```

[https://github.com/OneRingUSD/OneRing-Contracts/blob/30a7677e1eee63e26cde31bbcaf7d99d66097a2f/contracts/strategies/base/MasterChefBaseStrategy.sol#L216-L227]

```
    function _investAllAssets() internal {
        // address[] memory _vaultAssets = IVault(vault).underlyings();
        address _token0 = IUniswapV2Pair(underlying).token0();
        address _token1 = IUniswapV2Pair(underlying).token1();

        for (uint256 i = 0; i < 3; i++) {
            address _asset = IVault(vault).underlyings(i);
            address[] memory _route0 = fromVaultAssetRoutes[_asset][_token0];
            address[] memory _route1 = fromVaultAssetRoutes[_asset][_token1];
            _assetToUnderlying(_asset, _route0, _route1);
        }
```

```
}
```

[https://github.com/OneRingUSD/OneRing-Contracts/blob/30a7677e1eee63e26cde31bbcaf7d99d66097a2f/contracts/strategies/base/MiniChefStrategy.sol#L308-L319]

```
function initializeStakingRewardsBaseStrategy(
    address _vault,
    address _parentStrategy,
    address _underlying,
    address _rewardPool,
    address _router,
    address _rewardToken,
    address _dRewardToken,
    uint256 _sellFloor
) public initializer {
    __Ownable_init();
    vault = _vault;
    parentStrategy = _parentStrategy;

    underlying = _underlying;
    rewardPool = _rewardPool;
    router = _router;
    rewardToken = _rewardToken;
    dRewardToken = _dRewardToken;


    sellFloor = _sellFloor;
    sell = true;

    address _lpt;
    _lpt = IStakingRewards(_rewardPool).stakingToken();
    require(_lpt == underlying, "Pool Info doesn't match underlying");

    IERC20(underlying).safeApprove(rewardPool, 0);
    IERC20(underlying).safeApprove(rewardPool, uint256(-1));

    address _token0 = IUniswapV2Pair(underlying).token0();
    address _token1 = IUniswapV2Pair(underlying).token1();

    IERC20(_token0).safeApprove(router, 0);
    IERC20(_token0).safeApprove(router, uint256(-1));
```

```
        IERC20(_token1).safeApprove(router, 0);
        IERC20(_token1).safeApprove(router, uint256(-1));

        IERC20(underlying).safeApprove(router, 0);
        IERC20(underlying).safeApprove(router, uint256(-1));

        IERC20(rewardToken).safeApprove(router, 0);
        IERC20(rewardToken).safeApprove(router, uint256(-1));

        for (uint256 i = 0; i < 3; i++) {
            address _asset = IVault(vault).underlyings(i);
            IERC20(_asset).safeApprove(router, 0);
            IERC20(_asset).safeApprove(router, uint256(-1));
        }
    }
```

[https://github.com/OneRingUSD/OneRing-Contracts/blob/30a7677e1eee63e26cde31bbcaf7d99d66097a2f/contracts/strategies/base/StakingRewardsBaseStrategy.sol#L40-L91]

```
    function _investAllAssets() internal {
        // address[] memory _vaultAssets = IVault(vault).underlyings();
        address _token0 = IUniswapV2Pair(underlying).token0();
        address _token1 = IUniswapV2Pair(underlying).token1();

        for (uint256 i = 0; i < 3; i++) {
            address _asset = IVault(vault).underlyings(i);
            address[] memory _route0 = fromVaultAssetRoutes[_asset][_token0];
            address[] memory _route1 = fromVaultAssetRoutes[_asset][_token1];
            _assetToUnderlying(_asset, _route0, _route1);
        }
    }
```

[https://github.com/OneRingUSD/OneRing-Contracts/blob/30a7677e1eee63e26cde31bbcaf7d99d66097a2f/contracts/strategies/base/StakingRewardsBaseStrategy.sol#L216-L227]

Issue

The *MasterChefBaseStrategy#_investAllAssets()*, *MasterChefBaseStrategy#initializeMasterChefBaseStrategy()* *MiniChefStratyge #_investAllAssets()*, *StakingRewardsBaseStrategy#_investAllAssets()*, and *StakingRewardsBaseStrategy##initializeStakingRewardsBaseStrategy()*

functions traverse the vault's underlyings and perform specific actions. To control the loop used in this case, the constant 3 is used, not the length of the vault’s underlyings. This can trigger unintended behavior when the length of the vault's underlyings array is not 3.

Recommendation

We recommend adding a view function that returns the length of the vault’s underlyings to the vault contract and use it to control the loop statement.

Update

[v2.0] - OneRing team resolved the issue by changing constant(3) to length of the underlyings.

⚠ **MAJOR**

When changing the active strategy of OneRingVault contract, proper migration from the old strategy to the new strategy may fail to occur. (Found - v.1.0) (Resolved - v.2.0)

```
    function migrateStrategy(address _strategy, address _underlying)
        external
        onlyOwner
    {
        require(_underlying != address(0), "underlying must be defined");
        require(_strategy != address(0), "strategy must be defined");
        setActiveStrategy(_strategy);
        _withdrawAll(_underlying);
        _doHardWorkAll();
    }
```

[https://github.com/OneRingUSD/OneRing-Contracts/blob/30a7677e1eee63e26cde31bbcaf7d99d66097a2f/contracts/Vault.sol#L227-L235]

Issue

The *OneRingVault#migrateStrategy()* function changes the active strategy of the vault. At this time, the *OneRingVault#setActiveStrategy()* function is called before the *OneRingVault#_withdrawAll()* function, the balance is not moved from the old strategy to the new strategy. Thus, users cannot withdraw normally after the migration.

Recommendation

We advise changing the order of the functions *OneRingVault#setActiveStrategy()* and *OneRingVault#_withdrawAll()* in the *OneRingVault#migrateStrategy()* function.

Update

[v2.0] - OneRing team resolved the issue by *OneRingVault#migrateStrategy()* function logic.

⚠ **MAJOR**

When there is a disabled strategy during the operation of OneRingVault utilizing MultiStrategy, the amount deposited per strategy may be inaccurate. (Found - v.1.0)

```
function assetToUnderlying(address _inputAsset) public returns (uint256) {
    uint256 _totalDeposited;
    uint256 _length = strategyInfo.length;
    uint256 _inputBalance = IERC20(_inputAsset).balanceOf(address(this));

    if (_inputBalance == 0) {
        return 0;
    }

    for (uint256 _sid = 0; _sid < _length; _sid++) {
        StrategyInfo storage _strategyInfo = strategyInfo[_sid];

        if (strategyEnabled[_strategyInfo.strategy]) {
            uint256 _balanceToDeposit = _inputBalance
                .div(totalAllocPoint)
                .mul(_strategyInfo.allocPoint);

            if (_sid == _length - 1) {
                _balanceToDeposit = IERC20(_inputAsset).balanceOf(
                    address(this)
                );
            }

            IERC20(_inputAsset).safeTransfer(
                _strategyInfo.strategy,
                _balanceToDeposit
            );

            uint256 _added = IStrategy(strategyInfo[_sid].strategy)
                .assetToUnderlying(_inputAsset);

            uint256 _addedInUSD = IStrategy(strategyInfo[_sid].strategy)
                .getUSDBalanceFromUnderlyingBalance(_added);
            _totalDeposited = _totalDeposited.add(_addedInUSD);
        }
    }
```

```
        return _totalDeposited;
    }
```
[https://github.com/OneRingUSD/OneRing-Contracts/blob/30a7677e1eee63e26cde31bbcaf7d99d66097a2f/contracts/strategies/MultiStrategy.sol#L83-L121]

Issue

The *MultiStrategy#assetToUnderlying()* function deposits the assets Multistrategy possesses to the strategy registered in the contract as much as the ratio of the allocated allocPoint, respectively, out of all allocPoints.
However, when a disabled strategy exists, a small amount of assets is deposited to other enabled strategies because the disabled strategy is participating in the calculation of totalAllocPoint.

Recommendation

When a strategy is disabled, we recommend adding a statement that sets allocPoint of the strategy to 0 and modifies totalAllocPoint.

⚠ **MAJOR**

The withdrawToVault function withdraws abnormal values to the vault. (Found - v.1.0)

```
function withdrawToVault(uint256 _usdAmount, address _asset)
    external
    restricted
{
    address _token0 = IUniswapV2Pair(underlying).token0();
    address _token1 = IUniswapV2Pair(underlying).token1();
    uint256 _usdBalance = getUSDBalanceFromUnderlyingBalance(
        underlyingBalance()
    );
    uint256 _defaultUnit = uint256(10)**uint256(18);
    uint256 _assetUnit = uint256(10)**uint256(ERC20(_asset).decimals());

    uint256 _underlyingBal = IERC20(underlying).balanceOf(address(this));

    uint256 _amount = underlyingBalance().mul(_usdAmount).div(_usdBalance);

    if (_amount < _underlyingBal) {
        uint256 _toWithdraw = _amount.mul(_assetUnit).div(_defaultUnit);

        IERC20(_asset).safeTransferFrom(address(this), vault, _toWithdraw);

        return;
    }

    uint256 _missing = _amount.sub(_underlyingBal);
    if (_missing > rewardPoolUnderlyingBalance()) {
        _missing = rewardPoolUnderlyingBalance();
    }

    IMasterChef(masterChef).withdraw(poolId, _missing);

    _assetFromUnderlying(
        _asset,
        _missing,
        toVaultAssetRoutes[_asset][_token0],
        toVaultAssetRoutes[_asset][_token1]
    );

```

```
        IERC20(_asset).safeTransfer(
            vault,
            IERC20(_asset).balanceOf(address(this))
        );
    }
```

[https://github.com/OneRingUSD/OneRing-Contracts/blob/30a7677e1eee63e26cde31bbcaf7d99d66097a2f/contracts/strategies/base/MasterChefBaseStrategy.sol#L253-L295]

```
    function withdrawToVault(uint256 _usdAmount, address _asset)
        external
        restricted
    {
        address _token0 = IUniswapV2Pair(underlying).token0();
        address _token1 = IUniswapV2Pair(underlying).token1();
        uint256 _usdBalance = getUSDBalanceFromUnderlyingBalance(
            investedUnderlyingBalance()
        );
        uint256 _defaultUnit = uint256(10)**uint256(18);
        uint256 _assetUnit = uint256(10)**uint256(ERC20(_asset).decimals());

        uint256 _underlyingBal = IERC20(underlying).balanceOf(address(this));

        uint256 _amount = investedUnderlyingBalance().mul(_usdAmount).div(
            _usdBalance
        );

        if (_amount < _underlyingBal) {
            uint256 _toWithdraw = _amount.mul(_assetUnit).div(_defaultUnit);

            IERC20(_asset).safeTransferFrom(address(this), vault, _toWithdraw);

            return;
        }

        uint256 _missing = _amount.sub(_underlyingBal);
        if (_missing > rewardPoolBalance()) {
            _missing = rewardPoolBalance();
        }

        IMiniChef(miniChef).withdraw(poolId, _missing, address(this));
        _assetFromUnderlying(
            _asset,
            _missing,
            toVaultAssetRoutes[_asset][_token0],
            toVaultAssetRoutes[_asset][_token1]
        );

        IERC20(_asset).safeTransfer(
            vault,
            IERC20(_asset).balanceOf(address(this))
        );
    }
```

[https://github.com/OneRingUSD/OneRing-Contracts/blob/30a7677e1eee63e26cde31bbcaf7d99d66097a2f/contracts/strategies/base/MiniChefStrategy.sol#L345-L389]

```
    function withdrawToVault(uint256 _usdAmount, address _asset)
        external
        restricted
    {
        address _token0 = IUniswapV2Pair(underlying).token0();
        address _token1 = IUniswapV2Pair(underlying).token1();
        uint256 _usdBalance = getUSDBalanceFromUnderlyingBalance(
            underlyingBalance()
        );
        uint256 _defaultUnit = uint256(10)**uint256(18);
        uint256 _assetUnit = uint256(10)**uint256(ERC20(_asset).decimals());

        uint256 _underlyingBal = IERC20(underlying).balanceOf(address(this));

        uint256 _amount = underlyingBalance().mul(_usdAmount).div(_usdBalance);

        if (_amount < _underlyingBal) {
            uint256 _toWithdraw = _amount.mul(_assetUnit).div(_defaultUnit);

            IERC20(_asset).safeTransferFrom(address(this), vault, _toWithdraw);

            return;
        }

        uint256 _missing = _amount.sub(_underlyingBal);
        if (_missing > rewardPoolUnderlyingBalance()) {
            _missing = rewardPoolUnderlyingBalance();
        }

        IStakingRewards(rewardPool).withdraw(_missing);

        _assetFromUnderlying(
            _asset,
            _missing,
            toVaultAssetRoutes[_asset][_token0],
            toVaultAssetRoutes[_asset][_token1]
        );

        IERC20(_asset).safeTransfer(
            vault,
            IERC20(_asset).balanceOf(address(this))
        );
    }
```
[https://github.com/OneRingUSD/OneRing-Contracts/blob/30a7677e1eee63e26cde31bbcaf7d99d66097a2f/contracts/strategies/base/StakingRewardsBaseStrategy.sol#L253-L296]

Issue

The *MasterChefBaseStrategy#withdrawToVault()* and *MiniChefStrategy#withdrawToVault(),StakingRewardsBaseStrategy#withdrawToVault()* functions first swap the underlying asset of the strategy into an asset then transfer thereof to the vault.

However, _toWithdraw variable, which calculates in the case of _amount < _underlyingBal, is the unit of the underlying lpToken, not the unit of the asset. Therefore, abnormal values are sent to the vault. Also, in general, there is a high probability that there is no asset inside if the strategy does not call the _assetFromUnderlying function, the safeTransferFrom function inside the if statement is highly likely to fail in the case of _amount < _underlyingBal.

Recommendation

We advise modifying the strategy's underlying token to be swapped to asset token before transfer even if _amount < _underlyingBal.

MINOR

No syntax approves the asset to the router in the OneRingVault#addUnderlying() function. (Found - v.1.0)

```
function addUnderlying(address _underlying) public onlyOwner {
    require(_underlying != address(0), "_underlying must be defined");
    underlyings.push(_underlying);
    underlyingEnabled[_underlying] = true;
}
```

[https://github.com/OneRingUSD/OneRing-Contracts/blob/30a7677e1eee63e26cde31bbcaf7d99d66097a2f/contracts/Vault.sol#L206-L210]

Issue

The *OneRingVault#addUnderlying()* function adds a new address to the vault's underlyings. For the address added at this time, because the *IERC20(_asset).safeApprove(router, 0)* and *IERC20(_asset).safeApprove(router, uint256(-1))* functions are not called in the strategy, the deposit may not be completed normally.

Recommendation

We recommend calling the safeApprove function in active strategy when calling the *OneRingVault#addUnderlying()* function.

DISCLAIMER

This report does not guarantee investment advice, the suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects on main-net. In order to write secure smart contracts, correction of discovered problems and sufficient testing thereof are required.

Appendix A. Test Results

The following results show the unit test results covering the key logic of the smart contract subject to the security audit. Parts marked in red are test cases that failed to pass the test due to existing issues.

```
Vault
  using SpookyFantomMaiUsdcStrategy
    #initialize
      valid case
        ✓ set activeStrategy properly
        ✓ set underlyings properly
        ✓ set underlyingUnit properly
    #deposit
      ✓ should fail if try to deposit 0
      ✓ should fail if underlying is not enabled
      valid case
        1) get Vault share properly
        ✓ should emit Deposit event
    #depositFor
      ✓ should fail if try to deposit 0 (7585ms)
      ✓ should fail if beneficiary is zero address
      ✓ should fail if underlying is not enabled
      valid case
        ✓ should emit Deposit event (7355ms)
    #withdraw
      ✓ should fail if totalSupply is zero
      ✓ should fail if amount is zero (8035ms)
      ✓ should fail if underlying is not enabled (6368ms)
      valid case
        ✓ get underlying token properly
        ✓ should emit Withdraw event
    #withdrawAll
      ✓ should fail if msg.sender is not owner
      valid case
        ✓ should move underlying token to Vault
    #doHardWork
      ✓ should fail if msg.sender is not owner
      ✓ should fail amount is larger than available
      valid case
        ✓ should invest underlying token
        ✓ should liquidate reward token
        ✓ should emit Invest event
    #doHardWorkAll
      ✓ should fail if msg.sender is not owner
      valid case
        ✓ should invest underlying tokens
        ✓ should liquidate reward token
    #invest
      ✓ should fail if msg.sender is not owner
      ✓ should fail amount is larger than available
      valid case
        ✓ should invest underlying token
        ✓ should emit Invest event
    #addUnderlying
      ✓ should fail if msg.sender is not owner
      ✓ should fail if try to add zero Address
      valid case
        ✓ underlying added properly
        ✓ underlying enabled
      #enableUnderlying
        ✓ should fail if msg.sender is not owner
        ✓ should fail if try to enable zero Address
        valid case
          ✓ underlying enabled
      #disableUnderlying
        ✓ should fail if msg.sender is not owner
        ✓ should fail if try to enable zero Address
        valid case
          ✓ underlying disabled
      #setActiveStrategy
        ✓ should fail if msg.sender is not owner
        ✓ should fail if try to enable zero Address
        valid case
          ✓ set activeStrategy properly
      #migrateStrategy
        ✓ should fail if msg.sender is not owner
        ✓ should fail if try to migrate to zero Address
        ✓ should fail if underlying is zero Address
        valid case
          ✓ set activeStrategy properly
  migrate to SpookyFantomTusdUsdcStrategy
    2) withdraw properly after migration
    ✓ deposit to new strategy successfully (12760ms)
  MasterChefBaseStrategy Spec
```

```
   #setSellFloor
     ✓ should fail if msg.sender is not owner
     valid case
       ✓ set sellFloor properly
   #setParentStrategy
     ✓ should fail if msg.sender is not owner
     valid case
       ✓ set parentStrategy properly
   #setRouter
     ✓ should fail if msg.sender is not owner
     valid case
       ✓ set router properly
   #setVault
     ✓ should fail if msg.sender is not owner
     valid case
       ✓ set vault properly
   #emergencyExitRewardPool
     ✓ should fail if msg.sender is not owner
     valid case
       ✓ withdraw from reward pool properly
   #unsalvagableTokens
     ✓ get whether it is reward or underlying
   #salvage
     ✓ should fail if msg.sender is not owner
     ✓ should fail token is unsalvagable
     valid case
       ✓ transfer token properly

Vault
  using SushiPolygonTusdUsdcStrategy
   #initialize
     valid case
       ✓ set activeStrategy properly
       ✓ set underlyings properly
       ✓ set underlyingUnit properly
   #deposit
     ✓ should fail if try to deposit 0
     ✓ should fail if underlying is not enabled
     valid case
       3) get Vault share properly
       ✓ should emit Deposit event
   #depositFor
     ✓ should fail if try to deposit 0 (8716ms)
     ✓ should fail if beneficiary is zero address
```

```
    ✓ should fail if underlying is not enabled
    valid case
      ✓ should emit Deposit event (37764ms)
  #withdraw
    ✓ should fail if totalSupply is zero
    ✓ should fail if amount is zero (12879ms)
    ✓ should fail if underlying is not enabled (16218ms)
    valid case
      ✓ get underlying token properly
      ✓ should emit Withdraw event
  #withdrawAll
    ✓ should fail if msg.sender is not owner
    valid case
      ✓ should move underlying token to Vault
  #doHardWork
    ✓ should fail if msg.sender is not owner
    ✓ should fail amount is larger than available
    valid case
      ✓ should invest underlying token
      ✓ should liquidate reward token
      ✓ should emit Invest event
  #doHardWorkAll
    ✓ should fail if msg.sender is not owner
    valid case
      ✓ should invest underlying tokens
      ✓ should liquidate reward token
  #invest
    ✓ should fail if msg.sender is not owner
    ✓ should fail amount is larger than available
    valid case
      ✓ should invest underlying token
      ✓ should emit Invest event
  #addUnderlying
    ✓ should fail if msg.sender is not owner
    ✓ should fail if try to add zero Address
    valid case
      ✓ underlying added properly
      ✓ underlying enabled
    #enableUnderlying
      ✓ should fail if msg.sender is not owner
      ✓ should fail if try to enable zero Address
      valid case
        ✓ underlying enabled
    #disableUnderlying
```

```
          ✓ should fail if msg.sender is not owner
          ✓ should fail if try to enable zero Address
          valid case
            ✓ underlying disabled
        #setActiveStrategy
          ✓ should fail if msg.sender is not owner
          ✓ should fail if try to enable zero Address
          valid case
            ✓ set activeStrategy properly
        #migrateStrategy
          ✓ should fail if msg.sender is not owner
          ✓ should fail if try to migrate to zero Address
          ✓ should fail if underlying is zero Address
          valid case
            ✓ set activeStrategy properly
    migrate to SushiPolygonFraxUsdcStrategy
      4) withdraw properly after migration
      ✓ deposit to new strategy successfully
    MiniChefStrategy Spec
      #setSellFloor
        ✓ should fail if msg.sender is not owner
        valid case
          ✓ set sellFloor properly
      #setParentStrategy
        ✓ should fail if msg.sender is not owner
        valid case
          ✓ set parentStrategy properly
      #unsalvagableTokens
        ✓ get whether it is reward or underlying
      #salvage
        ✓ should fail if msg.sender is not owner
        ✓ should fail token is unsalvagable
        valid case
          ✓ transfer token properly

Vault
    using QuickPolygonUsdtUsdcStrategy
      #initialize
        valid case
          ✓ set activeStrategy properly
          ✓ set underlyings properly
          ✓ set underlyingUnit properly
      #deposit
        ✓ should fail if try to deposit 0
    ✓ should fail if underlying is not enabled
    valid case
      5) get Vault share properly
      ✓ should emit Deposit event
  #depositFor
    ✓ should fail if try to deposit 0
    ✓ should fail if beneficiary is zero address
    ✓ should fail if underlying is not enabled
    valid case
      ✓ should emit Deposit event (7973ms)
  #withdraw
    ✓ should fail if totalSupply is zero
    ✓ should fail if amount is zero (7685ms)
    ✓ should fail if underlying is not enabled (7706ms)
    valid case
      ✓ get underlying token properly
      ✓ should emit Withdraw event
    #withdrawAll
      ✓ should fail if msg.sender is not owner
      valid case
        ✓ should move underlying token to Vault
    #doHardWork
      ✓ should fail if msg.sender is not owner
      ✓ should fail amount is larger than available
      valid case
        ✓ should invest underlying token
        ✓ should liquidate reward token
        ✓ should emit Invest event
    #doHardWorkAll
      ✓ should fail if msg.sender is not owner
      valid case
        ✓ should invest underlying tokens
        ✓ should liquidate reward token
    #invest
      ✓ should fail if msg.sender is not owner
      ✓ should fail amount is larger than available
      valid case
        ✓ should invest underlying token
        ✓ should emit Invest event
    #addUnderlying
      ✓ should fail if msg.sender is not owner
      ✓ should fail if try to add zero Address
      valid case
        ✓ underlying added properly
```

```
          ✓ underlying enabled
        #enableUnderlying
          ✓ should fail if msg.sender is not owner
          ✓ should fail if try to enable zero Address
          valid case
            ✓ underlying enabled
        #disableUnderlying
          ✓ should fail if msg.sender is not owner
          ✓ should fail if try to enable zero Address
          valid case
            ✓ underlying disabled
        #setActiveStrategy
          ✓ should fail if msg.sender is not owner
          ✓ should fail if try to enable zero Address
          valid case
            ✓ set activeStrategy properly
    MasterChefBaseStrategy Spec
      #setSellFloor
        ✓ should fail if msg.sender is not owner
        valid case
          ✓ set sellFloor properly
      #setSell
        ✓ should fail if msg.sender is not owner
        valid case
          ✓ set sell properly
      #setParentStrategy
        ✓ should fail if msg.sender is not owner
        valid case
          ✓ set parentStrategy properly
      #setRouter
        ✓ should fail if msg.sender is not owner
        valid case
          ✓ set router properly
      #setVault
        ✓ should fail if msg.sender is not owner
        valid case
          ✓ set vault properly
      #unsalvagableTokens
        ✓ get whether it is reward or underlying
      #salvage
        ✓ should fail if msg.sender is not owner
        ✓ should fail token is unsalvagable
        valid case
          ✓ transfer token properly
```

```
Vault
  using MultiStrategy
    deposit to MultiStrategy
      ✓ allocate USD properly
    withdraw from MultiStrategy
      ✓ withdraw USD properly
    #addStrategy
      ✓ should fail if strategy is zero address
      ✓ should fail if strategy is already enabled
      ✓ should fail if msg.sender is not owner
      valid case
        ✓ add strategy properly
    #setStrategy
      ✓ should fail if msg.sender is not owner
      valid case
        ✓ set allocPoint properly
    #enableStrategy
      ✓ should fail if msg.sender is not owner
      ✓ should fail if strategy is zero address
      ✓ should fail if strategy is already enabled
      valid case
        ✓ set strategyEnabled properly
    #disableStrategy
      ✓ should fail if msg.sender is not owner
      ✓ should fail if strategy is zero address
      ✓ should fail if strategy is already disabled
      valid case
        ✓ set strategyEnabled properly
```

End of Document