yAudit Cozy safety module Review

Review Resources:

- Public Cozy documentation.
- Various private documentation notion files

Auditors:

- Panda
- HHK

Table of Contents

1 Review Summary

2 Scope

3 Code Evaluation Matrix

4 Findings Explanation

5 Critical Findings

6 High Findings

7 Medium Findings

1. Medium - Some users may not be able to complete their redemption because of a potential underflow

8 Low Findings

1. Low - User may deposit in a `triggered` module and get slashed as well as pay slightly more protocol fees

2. Low - Reset `lastSharedSafetyModuleUpdate_` when changing state similar to `lastConfigUpdate`

9 Gas Saving Findings

1. Gas - Do not cache immutable values

2. Gas - Inefficient Event Emission Order in acceptOwnership()

3. Gas - `_getCallerRole` can be re-written for more clarity

10 Informational Findings

1. Informational - Unused import

2. Informational - Typos

3. Informational - `else` block unnecessary

4. Informational - Rename `redemptionAmount_` parameter inside `updateRedemptionsAfterTrigger()`

5. Informational - Double check shared module compatibility in `setSharedSafetyModule()`

11 Final remarks

Review Summary

Cozy safety module

The Cozy Safety Module protocol makes it easy for project leaders (whether teams or DAOs) to set up and manage a safety module.

The contracts of the Cozy safety module Repo were reviewed over height days. Two auditors performed the code review between 21 April and 30 April 2025. The repository was under active development during the review, but the review was limited to the latest commit 638f6e740cedc07e8a96071fc628c792ef3710ab for the Cozy safety module repo.

Scope

The scope of the review consisted of the following contracts at the specific commit:

```
src/
├── CozySafetyModuleManager.sol
├── SafetyModule.sol
├── SafetyModuleFactory.sol
└── lib
    ├── Configurator.sol
    ├── ConfiguratorLib.sol
    ├── CozyMath.sol
    ├── Depositor.sol
    ├── FeesHandler.sol
```

```
    ├── Governable.sol
    ├── Ownable.sol
    ├── Redeemer.sol
    ├── RedemptionLib.sol
    ├── SafetyModuleBaseStorage.sol
    ├── SafetyModuleCalculationsLib.sol
    ├── SafetyModuleCommon.sol
    ├── SafetyModuleInspector.sol
    ├── SafetyModuleStates.sol
    ├── SlashHandler.sol
    ├── StateChanger.sol
    ├── StateTransitionsLib.sol
```

After the findings were presented to the Cozy safety module team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Cozy safety module and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Role-based permissions well implemented with a clear ownership model.
Mathematics	Average	Potential rounding issues in redemption calculations affecting invariants.
Complexity	Good	Clean architecture with well-defined responsibilities.
Libraries	Average	Some unnecessary imports and dependencies.
Decentralization	Average	Governance actors can significantly impact the safety module by pausing it or adding Trigger and Payout addresses, which reduces the level of decentralization.
Code stability	Good	The codebase remained stable, with little to no new development introduced during the review.
Documentation	Good	Inline comments and function documentation are present.
Monitoring	Good	Good event coverage.
Testing and verification	Average	Good test coverage and invariant suite, but some findings suggest room for improvement, such as integration tests.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas savings
  - Findings that can improve the gas efficiency of the contracts.
- Informational
  - Findings including recommendations and best practices.

Critical Findings

None.

High Findings

None.

Medium Findings

1. Medium - Some users may not be able to complete their redemption because of a potential underflow

Due to rounding errors, the `pool. pendingWithdrawalsAmount` may be smaller than a user's redemption amount, which could lead to underflow and block redemption completion.

Technical Details

The user can redeem from the safety module by calling `redeem()`, which will queue a redemption.
Later, they can call `completeRedemption()` to finalize the redemption and receive funds from the safety module.

When a slashing happens, the function `slash()` will first reduce the `reservePool_.pendingWithdrawalsAmount` (sum of all redemptions) by the percentage of slashing applied to the pool. Then it saves the percentage into the `pendingRedemptionAccISFs`.

When completing the redemption, the amount sent to the user will be recomputed using the `pendingRedemptionAccISFs` to apply any slashing during the redemption cooldown to the user.

The issue is that the `RedemptionLib` library in charge of computing the slashing over user's and pool's redemption may end up breaking the invariant `pool.pendingWithdrawalsAmount < sum(user's redemption_.totalAssetAmount)` due to a rounding errors.

This will lead to some last users not being able to complete their redemption as the function will underflow when subtracting their amount from

`reservePool_.pendingWithdrawalsAmount` inside `_completeRedemption()`.

POC:

```
function test_pendingUserNotGreaterThanPendingTotal(uint256 depositAmount_,
uint256 pendingWithdrawalsAmount_, uint256 slashAmount_, uint256 slashAmount2_)
public {
        pool.depositAmount = bound(depositAmount_, 100 ether, 500_000 ether);
        pool.pendingWithdrawalsAmount = bound(pendingWithdrawalsAmount_, 0.1
ether, pool.depositAmount); //can't withdraw more than deposit
        uint256 slashAmount1 = bound(slashAmount_, 0.1 ether, pool.depositAmount
- 0.1 ether); //can't slash more than deposit - min bound of second slash
        uint256 slashAmount2 = bound(slashAmount2_, 0.1 ether, pool.depositAmount
- slashAmount1); //can't slash more than deposit - first slash

        uint128 userPending = uint128(pool.pendingWithdrawalsAmount); //100% of
withdrawals

        //slash
        pool.pendingWithdrawalsAmount =
RedemptionLib.updateRedemptionsAfterTrigger(
            pool.pendingWithdrawalsAmount, pool.depositAmount, slashAmount1,
reservePoolPendingRedemptionAccISFs
        );
        pool.depositAmount -= slashAmount1;

        console.log("Inverted ratio added by first slash:");
        uint256 firstSlashRatio = reservePoolPendingRedemptionAccISFs[0];
        emit log_uint(reservePoolPendingRedemptionAccISFs[0]);

        //slash again
        pool.pendingWithdrawalsAmount =
RedemptionLib.updateRedemptionsAfterTrigger(
            pool.pendingWithdrawalsAmount, pool.depositAmount, slashAmount2,
reservePoolPendingRedemptionAccISFs
        );
        pool.depositAmount -= slashAmount2;

        uint256 newUserPending = RedemptionLib.computeFinalReserveAssetsRedeemed(
            reservePoolPendingRedemptionAccISFs, userPending, 1e18, 0
        );
```

```
        console.log("Inverted ratio added by second slash:");
        emit
log_uint(reservePoolPendingRedemptionAccISFs[0].divWadUp(firstSlashRatio));
        console.log("Inverted ratio after second slash:");
        emit log_uint(reservePoolPendingRedemptionAccISFs[0]);
        console.log("pool.pendingWithdrawalsAmount:");
        emit log_uint(pool.pendingWithdrawalsAmount);
        console.log("pool.depositAmount:");
        emit log_uint(pool.depositAmount);
        console.log("newUserPending:");
        emit log_uint(newUserPending);

        assertGe(pool.pendingWithdrawalsAmount, newUserPending); // WILL REVERT
    }
```

Impact

Medium. The invariant doesn't hold which can lead to underflow and dos redemption completion.

Recommendation

Round up the `pendingWithdrawalsAmount` by modifying `oldAssetsPendingRedemption_.mulWadDown(scalingFactor_);` to `oldAssetsPendingRedemption_.mulWadUp(scalingFactor_);` inside `computeNewPendingRedemptionsAccumulatedScalingFactor()`.

Developer Response

Fixed: https://github.com/Cozy-Finance/cozy-safety-module-private/pull/130.

Low Findings

1. Low - User may deposit in a `triggered` module and get slashed as well as pay slightly more protocol fees

The Safety module allows deposits even when the protocol is in `triggered` mode. This means a user unaware may deposit and have his deposit slashed without benefiting from rewards since he just deposited.

Additionally, the dripping system only drips fees when the vault is `active,` which means the user will pay slightly more fees than he should have.

Technical Details

The functions `depositReserveAssets()` and `depositReserveAssetsWithoutTransfer()` allow users to deposit into the safety module for a given reserve pool.

When depositing, the state of the safety module is checked; if it's `paused` it will revert. If it's `active` it will process normally, and if it's `triggered,` it will process normally except for not dripping the fees.

The risk is that a user unaware of the current state of the vault may deposit in a vault that was recently triggered. The vault could get triggered while the transaction is being broadcasted. Additionally, since the fees are not dripped, the user may end up paying more protocol fees than he should have.

Impact

Low. Users may deposit without knowing a slash is expected to happen very soon and pay extra fees.

Recommendation

- Add a parameter such as `bool allowTriggered` to the function and revert if the module is triggered so that users not aware of the incoming slashing are protected while still allowing deposits from the protocol and other actors that might still need to access the function.
- Update `reservePool_.lastFeesDripTime` to the current timestamp. While this might forfeit some of the fees for the Cozy protocol, it will protect new deposits from paying extra fees.

Developer Response

We acknowledge this functionality. Depositing while the SafetyModule remains triggered is designed to be used in special situations, not standard user deposits. For example, a protocol may want to use treasury funds to supplement SafetyModule reserve assets prior to slashing and payouts to reduce the impact on the other depositors. The Cozy frontend will alert and/or block users from depositing while a SafetyModule is triggered. The Cozy Safety Module docs will also make this functionality clear.

2. Low - Reset `lastSharedSafetyModuleUpdate_` when changing state similar to `lastConfigUpdate`

Technical Details

When the state changes from `triggered` to `active` inside `slash()` and from `active` to `paused` inside `pause()` the `lastConfigUpdate` is reset to `bytes32(0)`.

This protects the configuration change from accruing delay time while users cannot redeem their funds.

Similarly, when the shared safety module is updated, there is a delay before it goes into action. But during that delay if `slash()` or `pause()` are called, they won't reset the `lastSharedSafetyModuleUpdate_`.

This could be an issue since this will allow the update to accrue delay time and be completed as soon as the module goes back to `active` without users being able to withdraw.

While changing the shared safety module may not have as much impact as changing the configuration, the shared safety module is allowed to trigger the module without adding new trigger data. Thus, allowing users to redeem from the module is important before a new shared safety module is added.

Impact

Low.

Recommendation

Reset `lastSharedSafetyModuleUpdate_` when changing state similar to `lastConfigUpdate`.

Developer Response

Fixed: https://github.com/Cozy-Finance/cozy-safety-module-private/pull/128.

Gas Saving Findings

1. Gas - Do not cache immutable values

Caching immutable is not necessary.

Technical Details

```
File: src/CozySafetyModuleManager.sol

176: ISafetyModuleFactory safetyModuleFactory_ = safetyModuleFactory;
```

CozySafetyModuleManager.sol#L176

Impact

Gas savings.

Recommendation

Do not declare `safetyModuleFactory_` use `safetyModuleFactory` directly.

Developer Response

Fixed: https://github.com/Cozy-Finance/cozy-safety-module-private/pull/122.

2. Gas - Inefficient Event Emission Order in acceptOwnership()

The `acceptOwnership()` function creates an unnecessary local variable to store the old owner before emitting the event.

Technical Details

In the current implementation, a local variable `oldOwner_` stores the current owner before updating the state variable. This approach requires an extra SLOAD operation and local variable allocation.

```
File: Ownable.sol
address oldOwner_ = owner;
owner = msg.sender;
emit OwnershipTransferred(oldOwner_, msg.sender);
```

Ownable.sol#L43-L49

Impact

Gas savings.

Recommendation

```
delete pendingOwner;
emit OwnershipTransferred(owner, msg.sender);
owner = msg.sender;
```

Developer Response

Fixed: https://github.com/Cozy-Finance/cozy-safety-module-private/pull/126.

3. Gas - `_getCallerRole` can be re-written for more clarity

Technical Details

The `_getCallerRole` function currently uses a temporary variable `role_` to store the caller's role before returning it. This approach requires an unnecessary variable declaration and multiple assignments, resulting in higher gas costs.

```
File: StateChanger.sol
119:   function _getCallerRole(address who_) internal view returns (CallerRole) {
120:     CallerRole role_ = CallerRole.NONE;
121:     if (who_ == owner) role_ = CallerRole.OWNER;
122:     else if (who_ == pauser) role_ = CallerRole.PAUSER;
123:     // If the caller is the Manager itself, authorization for the call is
done
124:     // in the Manager.
125:     else if (who_ == address(cozySafetyModuleManager)) role_ =
CallerRole.MANAGER;
126:     return role_;
127:   }
```

Impact

Gas savings.

Recommendation

Refactor the function to use early returns instead of a temporary variable:

```
function _getCallerRole(address who_) internal view returns (CallerRole) {
  if (who_ == owner) return CallerRole.OWNER;
  if (who_ == pauser) return CallerRole.PAUSER;
  if (who_ == address(cozySafetyModuleManager)) return CallerRole.MANAGER;
```

```
    return CallerRole.NONE;
  }
```

Developer Response

Fixed: https://github.com/Cozy-Finance/cozy-safety-module-private/pull/127.

Informational Findings

1. Informational - Unused import

The identifier is imported but never used within the file.

Technical Details

```
File: src/CozySafetyModuleManager.sol

5: import {IERC20} from "cozy-safety-module-libs/interfaces/IERC20.sol";

11: import {UpdateConfigsCalldataParams, ReservePoolConfig} from
"./lib/structs/Configs.sol";

12: import {Delays} from "./lib/structs/Delays.sol";
```

src/CozySafetyModuleManager.sol#L5, src/CozySafetyModuleManager.sol#L11, src/CozySafetyModuleManager.sol#L12

```
File: src/SafetyModuleFactory.sol

4: import {IERC20} from "cozy-safety-module-libs/interfaces/IERC20.sol";

6: import {ReservePoolConfig} from "./lib/structs/Configs.sol";

7: import {Delays} from "./lib/structs/Delays.sol";
```

src/SafetyModuleFactory.sol#L4, src/SafetyModuleFactory.sol#L6, src/SafetyModuleFactory.sol#L7

```
File: src/lib/Configurator.sol

4: import {ICommonErrors} from "cozy-safety-module-
libs/interfaces/ICommonErrors.sol";

8: import {SafetyModuleState} from "./SafetyModuleStates.sol";

11: import {ISafetyModule} from "../interfaces/ISafetyModule.sol";

13: import {Delays} from "./structs/Delays.sol";

14: import {ReservePool} from "./structs/Pools.sol";

15: import {
16:   ConfigUpdateMetadata,
17:   ReservePoolConfig,
18:   UpdateConfigsCalldataParams,
19:   SharedSafetyModuleUpdateMetadata
20: } from "./structs/Configs.sol";

21: import {TriggerConfig} from "./structs/Trigger.sol";
```

src/lib/Configurator.sol#L4, src/lib/Configurator.sol#L8, src/lib/Configurator.sol#L11, src/lib/Configurator.sol#L13, src/lib/Configurator.sol#L14, src/lib/Configurator.sol#L15, src/lib/Configurator.sol#L21

```
File: src/lib/ConfiguratorLib.sol

5: import {IERC20} from "cozy-safety-module-libs/interfaces/IERC20.sol";

23: import {TriggerConfig, Trigger} from "./structs/Trigger.sol";
```

src/lib/ConfiguratorLib.sol#L5, src/lib/ConfiguratorLib.sol#L23

```
File: src/lib/FeesHandler.sol

6: import {IReceiptToken} from "cozy-safety-module-
libs/interfaces/IReceiptToken.sol";
```

```
12: import {ReservePool, AssetPool} from "./structs/Pools.sol";
```

src/lib/FeesHandler.sol#L6, src/lib/FeesHandler.sol#L12

```
File: src/lib/Redeemer.sol

4: import {ICommonErrors} from "cozy-safety-module-
libs/interfaces/ICommonErrors.sol";

5: import {IDripModel} from "cozy-safety-module-libs/interfaces/IDripModel.sol";

15: import {AssetPool, ReservePool} from "./structs/Pools.sol";
```

src/lib/Redeemer.sol#L4, src/lib/Redeemer.sol#L5, src/lib/Redeemer.sol#L15

```
File: src/lib/RedemptionLib.sol

9: import {ReservePool} from "./structs/Pools.sol";
```

src/lib/RedemptionLib.sol#L9

```
File: src/lib/SafetyModuleBaseStorage.sol

5: import {IReceiptToken} from "cozy-safety-module-
libs/interfaces/IReceiptToken.sol";
```

src/lib/SafetyModuleBaseStorage.sol#L5

```
File: src/lib/SafetyModuleCommon.sol

6: import {IERC20} from "cozy-safety-module-libs/interfaces/IERC20.sol";
```

src/lib/SafetyModuleCommon.sol#L6

```
File: src/lib/SlashHandler.sol
```

```
15: import {Trigger} from "./structs/Trigger.sol";
```

src/lib/SlashHandler.sol#L15

```
File: src/lib/StateChanger.sol

7: import {ISafetyModule} from "../interfaces/ISafetyModule.sol";
```

src/lib/StateChanger.sol#L7

Impact

Informational.

Recommendation

Clean up the code by removing unused import statements.

Developer Response

Fixed: https://github.com/Cozy-Finance/cozy-safety-module-private/pull/123.

2. Informational - Typos

Technical Details

```
File: src/lib/Redeemer.sol

// @audit: accomodate should be accommodate
95: // Fees were dripped already in this function if the SafetyModule is active,
so we don't need to accomodate for
```

src/lib/Redeemer.sol#L95

Impact

informational.

Recommendation

Fix the typo.

Developer Response

Fixed: https://github.com/Cozy-Finance/cozy-safety-module-private/pull/124.

3. Informational - `else` block unnecessary

By eliminating the `else` block and directly returning the values from the `if`-block, one level of nesting can be removed:

```
if (foo) {
  return 1;
} else {
  return 2;
}
```

Should be simplified to:

```
if (foo) return 1;
return 2;
```

Technical Details

```
File: src/lib/SafetyModuleInspector.sol

86: if (safetyModuleState == SafetyModuleState.ACTIVE) {
87:       return totalPoolAmount_
88:         - _getNextDripAmount(
89:           totalPoolAmount_,
90:
cozySafetyModuleManager.getFeeDripModel(ISafetyModule(address(this))),
91:           reservePool_.lastFeesDripTime
92:         );
93:     } else {
94:       return totalPoolAmount_;
95:     }
```

src/lib/SafetyModuleInspector.sol#L86

Impact

Informational.

Recommendation

Remove the else block.

Developer Response

Fixed: https://github.com/Cozy-Finance/cozy-safety-module-private/pull/125.

4. Informational - Rename `redemptionAmount_` parameter inside `updateRedemptionsAfterTrigger()`

Technical Details

The function `updateRedemptionsAfterTrigger()` has a parameter named `redemptionAmount_` however it is passed and used as `reservePool_.depositAmount`.

For this reason, it would be less confusing and more sensible to rename the variable to `depositAmount_`.

Impact

Informational.

Recommendation

Rename `redemptionAmount_` into `depositAmount_`.

Developer Response

Fixed: https://github.com/Cozy-Finance/cozy-safety-module-private/pull/129.

5. Informational - Double check shared module compatibility in `setSharedSafetyModule()`

Technical Details

When adding a shared module through `setSharedSafetyModule()` and `finalizeSetSharedSafetyModule()` the contract doesn't ensure that the contract passed is compatible and inherit all the functions needed on a shared safety module.

For safer configuration it could be beneficial to add an ERC165 support interface or a specific function such as `isSharedSafetyModule()` to the shared safety module interface that could be called inside `setSharedSafetyModule()` to ensure the contract passed is compatible.

Impact

Informational.

Recommendation

Add a function to identify the shared safety module and check it inside `setSharedSafetyModule()`.

Developer Response

Fix: https://github.com/Cozy-Finance/cozy-safety-module-private/pull/131.

Final remarks

The Cozy Safety Module presents a well-architected protocol with strong access controls and clean design principles. While the codebase is generally robust, it would be best to address the identified mathematics issues related to redemption calculations. Overall, the protocol demonstrates good engineering practices with room for minor improvements in testing coverage and redemption handling mechanics. The team provided extra fixes that were not directly linked to findings, and these commits were also reviewed, up to PR#132.