Zellic

**Prepared for**
**Payom Dousti**
Cozy Finance Inc.

**Prepared by**
**Jaeeu Kim**
**Jinseo Kim**
Zellic

**March 13, 2024**

# Safety Module

## Smart Contract Security Assessment

# Contents

## About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

# 1. Overview

## 1.1. Executive Summary

Zellic conducted a security assessment for Cozy Finance Inc. from Marth 5th to March 13th, 2024. During this engagement, Zellic reviewed Safety Module's code for security vulnerabilities, design issues, and general weaknesses in security posture.

## 1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Is there any issue with slashing and redemptions logic in Safety Module?
- Is there any issue with rounding/accounting logic in the Rewards Manager?
- Is there any issue that could lead to the loss of protocol/user funds?

## 1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Behavior of the CozyRouter contract
- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

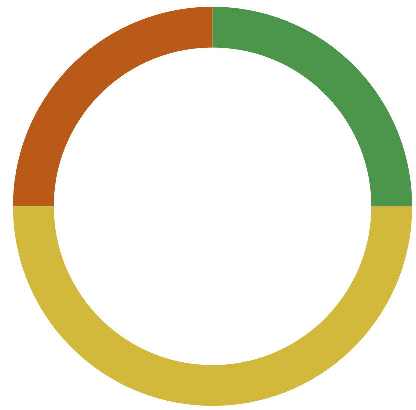
## 1.4. Results

During our assessment on the scoped Safety Module contracts, we discovered eight findings. No critical issues were found. Two findings were of high impact, four were of medium impact, and two were of low impact.

Additionally, Zellic recorded its notes and observations from the assessment for Cozy Finance Inc.'s benefit in the Discussion section (4. ↗) at the end of the document.

## Breakdown of Finding Impacts

| Impact Level | Count |
| --- | --- |
| Critical | 0 |
| High | 2 |
| Medium | 4 |
| Low | 2 |
| Informational | 0 |



# 2. Introduction

## 2.1. About Safety Module

Cozy Finance Inc. contributed the following description of Safety Module:

> The Cozy Safety Module is a way for protocols to deploy an on-chain, verifiable and credibly neutral safety module which protects users from edge cases like insolvent debt and smart contract hacks.

## 2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

**Basic coding mistakes.** Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

**Business logic errors.** Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

**Integration risks.** Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

**Code maturity.** We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood.

We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

## 2.3. Scope

The engagement involved a review of the following targets:

### Safety Module Contracts

| | |
|---|---|
| **Repositories** | https://github.com/Cozy-Finance/cozy-safety-module ↗<br>https://github.com/Cozy-Finance/cozy-safety-module-rewards-manager ↗<br>https://github.com/Cozy-Finance/cozy-safety-module-shared ↗ |
| **Versions** | cozy-safety-module: `fa15fdf4`<br>cozy-safety-module-rewards-manager: `ec4f4b55`<br>cozy-safety-module-shared: `08306753` |
| **Programs** | • CozySafetyModuleManager<br>• SafetyModule<br>• SafetyModuleFactory<br>• CozyManager<br>• RewardsManager<br>• RewardsManagerFactory<br>• StkReceiptToken<br>• ReceiptToken<br>• ReceiptTokenFactory |
| **Type** | Solidity |
| **Platform** | EVM-compatible |

## 2.4. Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of 2.2 person-weeks. The assessment was conducted over the course of 1.1 calendar weeks.

### Contact Information

The following project manager was associated with the engagement:

**Chad McDonald**
Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

**Jaeeu Kim**
Engineer
jaeeu@zellic.io

**Jinseo Kim**
Engineer
jinseo@zellic.io

## 2.5. Project Timeline

The key dates of the engagement are detailed below.

| | |
|---|---|
| **March 5, 2024** | Kick-off call |
| **March 5, 2024** | Start of primary review period |
| **March 13, 2024** | End of primary review period |

# 3. Detailed Findings

## 3.1. Deposit/stake functions are front-runnable

| | | | |
|---|---|---|---|
| **Target** | Depositor(SafetyModule), Depositor(RewardsManager), Staker | | |
| **Category** | Coding Mistakes | **Severity** | High |
| **Likelihood** | Medium | **Impact** | High |

### Description

Safety Module and Rewards Manager provide deposit/stake functions. These functions expect that transfer or approval has already been executed prior to a call.

In Depositor(SafetyModule), for example, the function `depositReserveAssets` is used to deposit with transfer. But, in this function, `safeTransferFrom` uses the arbitrary parameter `from_`, not `msg.sender`. In this case, if there is a user who has approved this contract, the other user could call this function by using `from_` as the approved address. If this function is not called with approval in a single transaction, it could be vulnerable to a front-run attack.

```
  function depositReserveAssets(uint8 reservePoolId_,
    uint256 reserveAssetAmount_, address receiver_, address from_)
    external
    returns (uint256 depositReceiptTokenAmount_)
  {
    // ...
! underlyingToken_.safeTransferFrom(from_, address(this),
    reserveAssetAmount_);

    depositReceiptTokenAmount_ =
      _executeReserveDeposit(reservePoolId_, underlyingToken_,
    reserveAssetAmount_, receiver_, assetPool_, reservePool_);
  }
```

### Impact

An attacker could execute a front-run attack when the user uses deposit/stake functions if `safeTransferFrom` is not called in a single transaction with approval.

In this case, the attacker could use `receiver_` as their address and `from_` as the victim's address, and the attacker could succeed in depositing the victim's tokens.

### Recommendations

We recommend changing `_from` to `msg.sender` in `depositReserveAssets`, `depositRewardAssets`, and `depositRewardAssets`. This could prevent an attacker from using other's approval even when the user does not use a single transaction.

### Remediation

Cozy Finance acknowledged the risk of front-running when not using a single transaction. Cozy Finance will provide router design that will have a multicall function for building the single transaction.

Cozy Finance fixed `depositReserveAssets`, `depositRewardAssets`, and `depositRewardAssets` for the users who do not use the multicall router.

This issue has been acknowledged by Cozy Finance Inc., and fixes were implemented in the following commits:

- 91aaaf63 ↗
- 375a19d4 ↗
- aa834308 ↗

### 3.2. Queued SafetyModule configuration updates may be applied when the SafetyModule is TRIGGERED by pausing.

| | | | |
|---|---|---|---|
| **Target** | ConfiguratorLib | | |
| **Category** | Coding Mistakes | **Severity** | High |
| **Likelihood** | Medium | **Impact** | High |

#### Description

In ConfiguratorLib, the function `finalizeUpdateConfigs` is used to update SafetyModule's settings. This function checks `safetyModuleState_ == SafetyModuleState.TRIGGERED`.

But this check could be bypassed by the owner. The owner of SafetyModule could call pause to transition the state from `TRIGGERED` to `PAUSED`. After that, the owner could call `finalizeUpdateConfigs` to update settings and then return back to `TRIGGERED` by calling `unpause`.

The state being `SafetyModuleState.TRIGGERED` means that the contract is ready to be slashed by using `maxSlashPercentage` and blocking the user's redeem.

However, in this case, `maxSlashPercentage` could be changed in `applyConfigUpdates`, which is called from `finalizeUpdateConfigs`. The user's deposit could be slashed more than they expect.

```
  function finalizeUpdateConfigs(
    // ...
  ) internal {
! if (safetyModuleState_ == SafetyModuleState.TRIGGERED)
    revert ICommonErrors.InvalidState();
    // ...
! applyConfigUpdates(reservePools_, triggerData_, delays_,
    receiptTokenFactory_, configUpdates_);
```

```
  function applyConfigUpdates(
    // ...
  ) public {
    // Update existing reserve pool maxSlashPercentages. Reserve pool assets
    cannot be updated.
    uint8 numExistingReservePools_ = uint8(reservePools_.length);
    for (uint8 i = 0; i < numExistingReservePools_; i++) {
! reservePools_[i].maxSlashPercentage
    = configUpdates_.reservePoolConfigs[i].maxSlashPercentage;
    }
    // ...
```

```
    }
```

## Impact

The user could check the risk of the pool's `maxSlashPercentage` before depositing and redeem before the update, because there is a delay for updating.

However, in this case, a malicious owner of SafetyModule could initialize the contract with a low `maxSlashPercentage`. Then, the owner could update the slash parameter to as high as 100% in `TRIGGERED`.

As a result of blocking the redeem in `TRIGGERED`, the user's balance is locked before updating and slashing. This could result in the user's deposit being slashed more than they expected by updating `maxSlashPercentage`.

## Recommendations

We recommend making the update queue zero when pausing the contract. This could prevent SafetyModule from updating in a triggered state.

## Remediation

This issue has been acknowledged by Cozy Finance Inc., and fixes were implemented in the following commits:

- `09bcbdf2` ↗
- `cda7786b` ↗
- `f22c62b9` ↗

## 3.3. SafetyModule Manager has full control of fee-dripping model

| | | | |
|---|---|---|---|
| **Target** | CozySafetyModuleManager, FeesHandler | | |
| **Category** | Business Logic | **Severity** | High |
| **Likelihood** | Low | **Impact** | Medium |

### Description

The protocol has the ability to collect fees from SafetyModules. The fee-dripping model of SafetyModule calculates the amount of fees. Specifically, invoking the `dripFactor()` function in the fee-dripping model contract, SafetyModule calculates the amount of assets that are dripped as fees.

The address of the fee-dripping–model contract is stored in CozySafetyModuleManager. The owner of CozySafetyModuleManager can change this address without limitation. The SafetyModule lacks the logic to cap the maximum amount of fees that can be dripped.

### Impact

If the owner account of CozySafetyModuleManager is compromised by a malicious actor, reserves of all SafetyModules can be stolen because the malicious actor can change the fee-dripping model and collect the entire reserves from SafetyModules as fees.

### Recommendations

Consider imposing limits on the changes to the fee-dripping model, such as capping the maximum fee that can be dripped, implementing a built-in time-lock mechanism for changing the fee-dripping model, or making the fee-dripping model immutable. Also, consider employing multi-signature and time-lock mechanisms for the owner account of the SafetyModule Manager.

### Remediation

This issue has been acknowledged by Cozy Finance Inc..

## 3.4. Rewards does not increase correctly due to rounding down

| | | | |
|---|---|---|---|
| **Target** | RewardsDistributor | | |
| **Category** | Coding Mistakes | **Severity** | Medium |
| **Likelihood** | Medium | **Impact** | Medium |

### Description

The Rewards Manager is responsible for distributing rewards to stakers. It calculates the total reward amount to be distributed over time.

```
function _getNextDripAmount(uint256 totalBaseAmount_, IDripModel dripModel_,
    uint256 lastDripTime_)
  internal
  view
  override
  returns (uint256)
{
  if (rewardsManagerState == RewardsManagerState.PAUSED) return 0;
  uint256 dripFactor_ = dripModel_.dripFactor(lastDripTime_);
  if (dripFactor_ > MathConstants.WAD) revert InvalidDripFactor();

  return _computeNextDripAmount(totalBaseAmount_, dripFactor_);
}

function _previewNextRewardDrip(RewardPool storage rewardPool_)
    internal view returns (RewardDrip memory) {
  return RewardDrip({
    rewardAsset: rewardPool_.asset,
    amount: _getNextDripAmount(rewardPool_.undrippedRewards,
    rewardPool_.dripModel, rewardPool_.lastDripTime)
  });
}

function _dripRewardPool(RewardPool storage rewardPool_) internal override {
  RewardDrip memory rewardDrip_ = _previewNextRewardDrip(rewardPool_);
  if (rewardDrip_.amount > 0) {
    rewardPool_.undrippedRewards -= rewardDrip_.amount;
    rewardPool_.cumulativeDrippedRewards += rewardDrip_.amount;
  }
  rewardPool_.lastDripTime = uint128(block.timestamp);
}
```

When Rewards Manager calculates the amount of rewards for a user, it internally updates the index snapshot, which is the total amount of reward divided by the total supply of staking receipt tokens.

```
// Round down, in favor of leaving assets in the pool.
uint256 unclaimedDrippedRewards_
    = cumulativeDrippedRewards_.mulDivDown(rewardsWeight_, MathConstants.ZOC)
  - claimableRewardsData_.cumulativeClaimableRewards;

nextClaimableRewardsData_.cumulativeClaimableRewards
    += unclaimedDrippedRewards_;
// Round down, in favor of leaving assets in the claimable reward pool.
nextClaimableRewardsData_.indexSnapshot
    += unclaimedDrippedRewards_.divWadDown(stkReceiptTokenSupply_);
```

In order to calculate the share of reward for a user the index snapshot is multiplied by the amount of staking receipt tokens. Once the reward is processed for a user, the current index snapshot is stored to the user information for managing the amount of claimed reward.

```
function _previewUpdateUserRewardsData(
  uint256 userStkReceiptTokenBalance_,
  uint256 newIndexSnapshot_,
  UserRewardsData storage userRewardsData_
) internal view returns (UserRewardsData memory newUserRewardsData_) {
  newUserRewardsData_.accruedRewards = userRewardsData_.accruedRewards
    + _getUserAccruedRewards(userStkReceiptTokenBalance_, newIndexSnapshot_,
    userRewardsData_.indexSnapshot);
  newUserRewardsData_.indexSnapshot = newIndexSnapshot_;
}

function _getUserAccruedRewards(
  uint256 stkReceiptTokenAmount_,
  uint256 newRewardPoolIndex,
  uint256 oldRewardPoolIndex
) internal pure returns (uint256) {
  // Round down, in favor of leaving assets in the rewards pool.
  return stkReceiptTokenAmount_.mulWadDown(newRewardPoolIndex
    - oldRewardPoolIndex);
}
```

If the decimal of the staking receipt token, which is of the staked token, is greater than the decimal of the reward token, the total reward for a short period, such as seconds, can become zero when divided by the total supply of the staking receipt token.

Assume the TKN18 token, which has 18 decimals, is staked, and the TKN9 token, which has 9 decimals, is rewarded. Suppose 10,000 TKN18 is staked in the Rewards Manager, and the total reward for a year is 100 TKN9. The total reward for three seconds would be slightly less than 0.00001 TKN9.

This reward amount is rounded down to zero when divided by the total supply of the staking receipt token. Therefore, if the index-snapshot updating logic is invoked three seconds after it was invoked last time, the index snapshot will not increase for that period of three seconds.

Exploiting this, an attacker can invoke the index-snapshot updating logic very frequently in order to prevent the index snapshot from increasing.

## Impact

An attacker can prevent users from claiming rewards accrued during the attack.

## Recommendations

Consider refactoring the reward logic or improving the accuracy of the index-snapshot calculation.

## Remediation

This issue has been acknowledged by Cozy Finance Inc., and a fix was implemented in commit 2a2c49c0 ↗.

## 3.5. Fees could be dripped in any state

| | | | |
|---|---|---|---|
| **Target** | FeesHandler, Depositor, Redeemer | | |
| **Category** | Coding Mistakes | **Severity** | Medium |
| **Likelihood** | Low | **Impact** | Medium |

### Description

According to Cozy's design, fees could be dripped when SafetyModules's state is `SafetyModuleState.ACTIVE`.

In FeesHandler, the function `_dripFeesFromReservePool` is used to accumulate dripped fees from the reserve pool. This function is called from `dripFees`, `dripFeesFromReservePool`, `claimFees`, `_executeReserveDeposit`, and `redeem`.

But only two functions — `dripFees` and `dripFeesFromReservePool` — check state. The others do not check that SafetyModules's state is `SafetyModuleState.ACTIVE`.

This causes fees to drip in any state which means that the fees may be overcharged.

```
  function dripFees() public override {
!  if (safetyModuleState != SafetyModuleState.ACTIVE) return;
    IDripModel dripModel_
    = cozySafetyModuleManager.getFeeDripModel(ISafetyModule(address(this)));

    uint256 numReserveAssets_ = reservePools.length;
    for (uint8 i = 0; i < numReserveAssets_; i++) {
!  _dripFeesFromReservePool(reservePools[i], dripModel_);
    }
  }

  function dripFeesFromReservePool(uint8 reservePoolId_) external {
!  if (safetyModuleState != SafetyModuleState.ACTIVE) return;
    IDripModel dripModel_
    = cozySafetyModuleManager.getFeeDripModel(ISafetyModule(address(this)));

!  _dripFeesFromReservePool(reservePools[reservePoolId_], dripModel_);
  }

  function claimFees(address owner_) external {
    // ...
```

```
    uint256 numReservePools_ = reservePools.length;
    for (uint8 i = 0; i < numReservePools_; i++) {
      ReservePool storage reservePool_ = reservePools[i];
!  _dripFeesFromReservePool(reservePool_, dripModel_);

      // ...
    }
  }

!  function _dripFeesFromReservePool(ReservePool storage reservePool_,
    IDripModel dripModel_) internal override {
    uint256 drippedFromDepositAmount_ = _getNextDripAmount(
      reservePool_.depositAmount - reservePool_.pendingWithdrawalsAmount,
    dripModel_, reservePool_.lastFeesDripTime
    );

    if (drippedFromDepositAmount_ > 0) {
      reservePool_.feeAmount += drippedFromDepositAmount_;
      reservePool_.depositAmount -= drippedFromDepositAmount_;
    }

    reservePool_.lastFeesDripTime = uint128(block.timestamp);
  }
```

## Impact

The function `_dripFeesFromReservePool` subtracts fees from `reservePool_.depositAmount`. This means a decrease in the asset price of depositing users.

A malicious user could call `claimFees` repeatedly no matter SafetyModule's state. This causes unexpected fees to be charged from SafetyModule.

## Recommendations

We recommend adding the check for `SafetyModuleState.ACTIVE` to all points before calling `_dripFeesFromReservePool`. This could prevent SafetyModule from dripping fees when the state is not active.

## Remediation

This issue has been acknowledged by Cozy Finance Inc., and fixes were implemented in the following commits:

- 1ffc3984 ↗

- a3cdd300
- 7e6909c8

## 3.6. Wrong ERC in reserve pool could brick fee system

| | | | |
|---|---|---|---|
| **Target** | FeesHandler | | |
| **Category** | Coding Mistakes | **Severity** | Medium |
| **Likelihood** | Low | **Impact** | Medium |

### Description

In FeesHandler, the function `claimFees` is used to claim fees from `cozySafetyModuleManager`. This function always iterates all reserve pools. But if one reserve pool causes the call to revert then `claimFees` always fails.

In this case, since there is no function to collect fees from each reserve pool, the fees become trapped in the SafetyModule.

```
function claimFees(address owner_) external {
  // Cozy fee claims will often be batched, so we require it to be initiated
    from the CozySafetyModuleManager to save
  // gas by removing calls and SLOADs to check the owner addresses each time.
  if (msg.sender != address(cozySafetyModuleManager))
    revert Ownable.Unauthorized();
  IDripModel dripModel_
    = cozySafetyModuleManager.getFeeDripModel(ISafetyModule(address(this)));

  uint256 numReservePools_ = reservePools.length;
  for (uint8 i = 0; i < numReservePools_; i++) {
    ReservePool storage reservePool_ = reservePools[i];
    _dripFeesFromReservePool(reservePool_, dripModel_);

    uint256 feeAmount_ = reservePool_.feeAmount;
    if (feeAmount_ > 0) {
      IERC20 asset_ = reservePool_.asset;
      reservePool_.feeAmount = 0;
      assetPools[asset_].amount -= feeAmount_;
      asset_.safeTransfer(owner_, feeAmount_);

      emit ClaimedFees(asset_, feeAmount_, owner_);
    }
  }
}
```

## Impact

A malicious owner of the SafetyModule could create incorrect ERC tokens and add them to the reserve-pool list. As a result, the `cozySafetyModuleManager` would fail to collect fees from the SafetyModule.

## Recommendations

We recommend adding a function to collect from each reserve pool. This measure would help prevent incorrect ERC tokens from obstructing the fee collection process.

## Remediation

This issue has been acknowledged by Cozy Finance Inc., and fixes were implemented in the following commits:

- `92c6032f` ↗
- `a7dfea8a` ↗

### 3.7. Partial deprivileging of the SafetyModule's manager

| Target | Governable, StateChanger | | |
|---|---|---|---|
| Category | Coding Mistakes | Severity | Low |
| Likelihood | Low | Impact | Low |

## Description

In Governable, the function `updatePauser` is used to update SafetyModule's pauser. However, this function does not have any checks, so `pauser` could be set to `cozySafetyModuleManager`, which has the role of `CallerRole.MANAGER`.

In StateChanger, the function `_getCallerRole` is used to check the role of the address. When `who_` is `pauser`, it returns `CallerRole.PAUSER` even though `who_` is the `cozySafetyModuleManager` because of the if-else routine.

```
function updatePauser(address _newPauser) external {
  if (msg.sender != owner && msg.sender != pauser) revert Unauthorized();
  emit PauserUpdated(_newPauser);
  pauser = _newPauser;
```

```
function _getCallerRole(address who_) internal view returns (CallerRole) {
  CallerRole role_ = CallerRole.NONE;
  if (who_ == owner) role_ = CallerRole.OWNER;
  else if (who_ == pauser) role_ = CallerRole.PAUSER;
  // If the caller is the Manager itself, authorization for the call is done
  // in the Manager.
  else if (who_ == address(cozySafetyModuleManager)) role_
    = CallerRole.MANAGER;
  return role_;
}
```

## Impact

The role of `CallerRole.MANAGER` has more permissions than `CallerRole.PAUSER`. This means that the manager cannot access functions before updating `pauser` to the other address.

### Recommendations

We recommend adding a check that `_newPauser` is `cozySafetyModuleManager` to prevent the manager from being deprivileged.

### Remediation

This issue has been acknowledged by Cozy Finance Inc., and a fix was implemented in commit d0112ef4 ↗.

## 3.8. Deterministic address could be used in front-run

| | | | |
|---|---|---|---|
| **Target** | SafetyModuleFactory, Rewards-ManagerFactory | | |
| **Category** | Coding Mistakes | **Severity** | Low |
| **Likelihood** | Low | **Impact** | Low |

### Description

SafetyModule and RewardsManager are deployed from the factory contract. In this process, the factory contract uses the deterministic contract address using `salt`.

However, `salt` does not include the caller's data like `msg.sender` does. In this case, providing the same `salt` as the input could make the same predicted contract address which could be vulnerable to a front-run attack.

```
  function createSafetyModule(
    address owner_,
    address pauser_,
    UpdateConfigsCalldataParams calldata configs_,
!   bytes32 salt_
  ) external returns (ISafetyModule safetyModule_) {
    // ...
    ISafetyModuleFactory safetyModuleFactory_ = safetyModuleFactory;
    isSafetyModule[ISafetyModule(safetyModuleFactory_.computeAddress(salt_))]
    = true;
!   safetyModule_ = safetyModuleFactory_.deploySafetyModule(owner_, pauser_,
    configs_, salt_);
  }
```

```
  function deploySafetyModule(
    address owner_,
    address pauser_,
    UpdateConfigsCalldataParams calldata configs_,
!   bytes32 baseSalt_
  ) public returns (ISafetyModule safetyModule_) {
    // ...
!   safetyModule_ =
    ISafetyModule(address(safetyModuleLogic).cloneDeterministic(salt(baseSalt_)));
    emit SafetyModuleDeployed(safetyModule_);
    safetyModule_.initialize(owner_, pauser_, configs_);
```

```
  function computeAddress(bytes32 baseSalt_) external view returns (address) {
    return Clones.predictDeterministicAddress(address(rewardsManagerLogic),
    salt(baseSalt_), address(this));
  }

  function salt(bytes32 baseSalt_) public view returns (bytes32) {
    // ...
!  return keccak256(abi.encode(baseSalt_, block.chainid));
  }
```

## Impact

An attacker could front-run deploying functions. If a user of the factory contract does not check for deploying transaction success and interacts with the contract using an address from `computeAddress` in the factory contract, the user could use a malicious contract.

For example, an attacker could silently add their payout address to change the direction of money movement.

## Recommendations

We recommend adding caller (`msg.sender`) data to `salt`. It could prevent an attacker from deploying to the same contract address that is expected by the user.

## Remediation

This issue has been acknowledged by Cozy Finance Inc., and fixes were implemented in the following commits:

- 7450fb85 ↗
- bd86c210 ↗

# 4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

## 4.1. CozySafetyModuleManager could pause/unpause each SafetyModule

The protocol manager has the ability to pause or unpause each module without requiring the agreement of the module's owner. We recommend clearly documenting this in the development documentation.

# 5. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

Please note that our threat model was based on the following commits:

- fa15fdf4 ↗
- ec4f4b55 ↗
- 08306753 ↗

The commits above represent a specific snapshot of the codebase. Therefore, it's important to understand that the absence of certain tests in our report may not reflect the current state of the test suite.

During the remediation phase, Cozy Finance Inc. took proactive steps to address the findings by adding test cases where applicable. This demonstrates their dedication to enhancing the code quality and overall reliability of the system, which is commendable.

## 5.1. Module: Configurator.sol

### Function: `updateConfigs(StakePoolConfig[] stakePoolConfigs_, RewardPoolConfig[] rewardPoolConfigs_)`

This function is used to execute a config update to the Rewards Manager. The caller must be the owner, and `ConfiguratorLib.updateConfigs` is called to update the stake pools and reward pools.

### Inputs

- `stakePoolConfigs_`
  - **Control**: Arbitrary.
  - **Constraints**: Validated in the `ConfiguratorLib.updateConfigs` function.
  - **Impact**: Array of new stake-pool configurations.
- `rewardPoolConfigs_`
  - **Control**: Arbitrary.
  - **Constraints**: Validated in the `ConfiguratorLib.updateConfigs` function.
  - **Impact**: Array of new reward-pool configurations.

### Branches and code coverage

**Intended branches**

- Updates the configuration accordingly.
    - ☑ Test coverage

**Negative behavior**

- Revert if the asset of existing stake pool is changed.
    - ☑ Negative test
- Revert if multiple stake pools refer to the same asset.
    - ☑ Negative test
- Revert if the asset of existing reward pool is changed.
    - ☑ Negative test
- Revert if assets of new stake pools are not strictly sorted.
    - ☑ Negative test
- Revert if the sum of reward weights is not 10,000.
    - ☑ Negative test

## 5.2. Module: CozyManager.sol

### Function: `createRewardsManager(address owner_, address pauser_, StakePoolConfig[] stakePoolConfigs_, RewardPoolConfig[] rewardPoolConfigs_, byte[32] salt_)`

This function is used to deploy a new RewardsManager with the provided parameters. This function calls `deployRewardsManager` from the rewardsManagerFactory contract to deploy a new RewardsManager.

### Inputs

- `owner_`
    - **Control**: Arbitrary.
    - **Constraints**: Not zero.
    - **Impact**: Address of the owner.
- `pauser_`
    - **Control**: Arbitrary.
    - **Constraints**: Not zero.
    - **Impact**: Address of the pauser.
- `stakePoolConfigs_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Array of stake pool configs.
- `rewardPoolConfigs_`
    - **Control**: Arbitrary.
    - **Constraints**: None.

    - **Impact**: Array of reward pool configs.
- `salt_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Value used to compute the resulting address of the RewardsManager.

## Branches and code coverage

**Intended branches**

- Create a new RewardsManager module.
    - ☑ Test coverage

**Negative behavior**

- Revert if the owner is zero.
    - ☑ Negative test
- Revert if the pauser is zero.
    - ☑ Negative test

## Function: `pause(IRewardsManager[] rewardsManagers_)`

This function is used to pause the Rewards Managers in the `rewardsManagers_` array. The Cozy-Manager's pauser or owner can perform this action.

## Inputs

- `rewardsManagers_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Array of Rewards Managers to pause.

## Branches and code coverage

**Intended branches**

- Pause the Rewards Managers in the given input array.
    - ☑ Test coverage

**Negative behavior**

- Revert if the caller is not the pauser or owner.
    - ☑ Negative test

### Function: `unpause(IRewardsManager[] rewardsManagers_)`

This function is used to unpause the Rewards Managers in the `rewardsManagers_` array. The owner of CozyManager can perform this action.

#### Inputs

- `rewardsManagers_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Array of Rewards Managers to unpause.

#### Branches and code coverage

**Intended branches**

- Unpause the Rewards Managers in the given input array.
    - ☑ Test coverage

**Negative behavior**

- Revert if the caller is not the owner.
    - ☑ Negative test

## 5.3. Module: Depositor.sol

### Function: `depositRewardAssetsWithoutTransfer(uint16 rewardPoolId_, uint256 rewardAssetAmount_, address receiver_)`

This fuction is used to deposit `rewardAssetAmount_` assets into the `rewardPoolId_` reward pool and mint `depositReceiptTokenAmount_` tokens to `receiver_`. It assumes that the user has already transferred `rewardAssetAmount_` of the reward pool's asset to the Rewards Manager; `_executeRewardDeposit` is called to execute the deposit and mint the deposit-receipt tokens.

#### Inputs

- `rewardPoolId_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Value of the reward-pool ID.
- `rewardAssetAmount_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Amount of the reward pool's asset to deposit.

- `receiver_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Address to mint the deposit-receipt tokens to.

## Branches and code coverage

**Intended branches**

- Increment the internal asset amount.
    - ☑ Test coverage
- Mint the deposit-receipt tokens.
    - ☑ Test coverage

**Negative behavior**

- Revert if the Rewards Manager is paused.
    - ☑ Negative test
- Revert if the amount that gets transferred is less than the amount to be deposited.
    - ☑ Negative test
- Revert if the result of the deposit is zero.
    - ☑ Negative test

## Function: `depositRewardAssets(uint16 rewardPoolId_, uint256 rewardAssetAmount_, address receiver_, address from_)`

This function is used to deposit `rewardAssetAmount_` assets into the `rewardPoolId_` reward pool on behalf of `from_` and mint `depositReceiptTokenAmount_` tokens to `receiver_`. It assumes that `from_` has approved the Rewards Manager to spend `rewardAssetAmount_` of the reward pool's asset; `_executeRewardDeposit` is called to execute the deposit and mint the deposit-receipt tokens.

## Inputs

- `rewardPoolId_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Value of the reward-pool ID.
- `rewardAssetAmount_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Amount of the reward pool's asset to deposit.
- `receiver_`
    - **Control**: Arbitrary.

- **Constraints**: None.
- **Impact**: Address to mint the deposit-receipt tokens to.

- `from_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address to pull the reward pool's asset from.

## Branches and code coverage

**Intended branches**

- Transfer the asset from the caller.
  - ☑ Test coverage
- Increment the internal asset amount.
  - ☑ Test coverage
- Mint the deposit-receipt tokens.
  - ☑ Test coverage

**Negative behavior**

- Revert if the Rewards Manager is paused.
  - ☑ Negative test
- Revert if the amount that gets transferred is less than the amount to be deposited.
  - ☑ Negative test
- Revert if the result of the deposit is zero.
  - ☑ Negative test

## Function: `previewUndrippedRewardsRedemption(uint16 rewardPoolId_, uint256 depositReceiptTokenAmount_)`

This function is used to preview the amount of undripped rewards that can be redeemed for `depositReceiptTokenAmount_` from a given reward pool; `_previewRedemption` is called to preview the amount of undripped rewards that can be redeemed.

## Inputs

- `rewardPoolId_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the reward-pool ID.
- `depositReceiptTokenAmount_`
  - **Control**: Arbitrary.
  - **Constraints**: None.

- **Impact**: Amount of deposit-receipt tokens to redeem.

### Branches and code coverage

**Intended branches**

- Returns the amount of undripped rewards.
  - [x] Test coverage

### Function: `redeemUndrippedRewards(uint16 rewardPoolId_, uint256 depositReceiptTokenAmount_, address receiver_, address owner_)`

This function is used to redeem by burning `depositReceiptTokenAmount_` of `rewardPoolId_` reward-pool deposit-receipt tokens and sending `rewardAssetAmount_` of `rewardPoolId_` reward-pool assets to `receiver_`. Reward-pool assets can only be redeemed if they have not been dripped yet. Also, `_previewRedemption` is called to preview the amount of undripped rewards that can be redeemed. The deposit-receipt tokens are burned and the reward-pool accounting is updated. The reward pool's asset is transferred to the receiver.

### Inputs

- `rewardPoolId_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the reward-pool ID.
- `depositReceiptTokenAmount_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Amount of deposit-receipt tokens to burn.
- `receiver_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address to send the reward pool's asset to.
- `owner_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the owner of the deposit-receipt tokens.

### Branches and code coverage

**Intended branches**

- Drip fees if the Rewards Manager is active.
    - ☑ Test coverage
- Burn the deposit-receipt tokens.
    - ☑ Test coverage
- Subtract the the internal asset amount.
    - ☑ Test coverage

**Negative behavior**

- Revert if the amount of the reward pool's asset to redeem is zero.
    - ☑ Negative test

## 5.4. Module: RewardsDistributor.sol

### Function: `claimRewards(uint16[] stakePoolIds_, address receiver_)`

This function is used to claim rewards for a set of stake pools and transfer rewards to `receiver_`. The function `claimRewards` calls `_claimRewards` with the given arguments; `_claimRewards` is an internal function that is used to claim rewards for a set of stake pools and transfer rewards to `receiver_`.

### Inputs

- `stakePoolIds_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Value of the stake-pool IDs.
- `receiver_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Address to transfer the claimed rewards to.

### Branches and code coverage

**Intended branches**

- Calls `_claimRewards` to update caller's reward and transfer reward tokens to receiver.
    - ☑ Test coverage

### Function: `claimRewards(uint16 stakePoolId_, address receiver_)`

This function is used to claim rewards for a specific stake pool and transfer rewards to `receiver_`. The function `claimRewards` calls `_claimRewards` with the given arguments; `_claimRewards` is an internal function that is used to claim rewards for a set of stake pools and transfer rewards to `re-`

`ceiver_`.

## Inputs

- `stakePoolId_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Value of the stake-pool ID.
- `receiver_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Address to transfer the claimed rewards to.

## Branches and code coverage

**Intended branches**

- Calls `_claimRewards` with a stake pool to update caller's reward and transfer reward tokens to receiver.
    - ☑ Test coverage

## Function: `dripRewardPool(uint16 rewardPoolId_)`

This function is used to drip rewards for a specific reward pool.

## Inputs

- `rewardPoolId_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Value of the reward-pool ID.

## Branches and code coverage

**Intended branches**

- Calls `_dripRewardPool` to update the dripped amount of the reward pool with the ID `rewardPoolId_`.
    - ☑ Test coverage

**Negative behavior**

- Revert if the Rewards Manager is paused.

- ☑ Negative test

### Function: `dripRewards()`

This function is used to drip rewards for all reward pools.

#### Branches and code coverage

**Intended branches**

- Calls `_dripRewardPool` to update the dripped amounts of all reward pools.
  - ☑ Test coverage

**Negative behavior**

- Revert if the Rewards Manager is paused.
  - ☑ Negative test

### Function: `previewClaimableRewards(uint16[] stakePoolIds_, address owner_)`

This function is used to preview the claimable rewards for a given set of stake pools. Also, `previewClaimableRewards` calls `_previewClaimableRewards` with the given arguments; `_previewClaimableRewards` is an internal function that is used to preview the claimable rewards for a given set of stake pools.

#### Inputs

- `stakePoolIds_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the stake-pool IDs.
- `owner_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the user to preview claimable rewards for.

#### Branches and code coverage

**Intended branches**

- Returns the claimable rewards for a given set of stake pools and claimer.
  - ☑ Test coverage

#### Function: `updateUserRewardsForStkReceiptTokenTransfer(address from_, address to_)`

This function is used to update the user-rewards data to prepare for a transfer of stkReceiptTokens. Also, `updateUserRewardsForStkReceiptTokenTransfer` calls `_updateUserRewards` with the given arguments; `_updateUserRewards` is an internal function that is used to update the user-rewards data to prepare for a transfer of stkReceiptTokens.

This function is called from the `transfer` function of the StkReceiptToken contract. It updates the user's rewards before transferring the stkReceiptTokens by calling into the Rewards Manager.

#### Inputs

- `from_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the user transferring stkReceiptTokens.
- `to_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the user receiving stkReceiptTokens.

#### Branches and code coverage

**Intended branches**

- Calls `_updateUserRewards` to update `from`'s reward.
  - ☑ Test coverage
- Calls `_updateUserRewards` to update `to`'s reward.
  - ☑ Test coverage

**Negative behavior**

- Revert if the caller is not a registered stkReceiptToken in the Rewards Manager.
  - ☑ Negative test

### 5.5. Module: RewardsManagerFactory.sol

#### Function: `computeAddress(byte[32] baseSalt_)`

This function is used to compute and return the address that the RewardsManager will be deployed to.

## Inputs

- `baseSalt_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value used to compute the resulting address of the RewardsManager.

## Branches and code coverage

**Intended branches**

- Compute the RewardsManager's address correctly from the given base salt.
  - [x] Test coverage

## Function: `deployRewardsManager(address owner_, address pauser_, StakePoolConfig[] stakePoolConfigs_, RewardPoolConfig[] rewardPoolConfigs_, byte[32] baseSalt_)`

This function is used to deploy a new RewardsManager contract with the specified configuration.

## Inputs

- `owner_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the owner of the RewardsManager.
- `pauser_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the pauser of the RewardsManager.
- `stakePoolConfigs_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Array of stake pool configurations.
- `rewardPoolConfigs_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Array of reward pool configurations.
- `baseSalt_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value used to compute the resulting address of the RewardsManager.

### Branches and code coverage

**Intended branches**

- Deploy the RewardsManager module.
  - ☑ Test coverage
- Initialize the deployed RewardsManager module.
  - ☑ Test coverage

**Negative behavior**

- Revert if the caller is not the Cozy manager.
  - ☑ Negative test

## 5.6. Module: RewardsManagerInspector.sol

### Function: `convertRewardAssetToReceiptTokenAmount(uint16 rewardPoolId_, uint256 rewardAssetAmount_)`

This function is used to convert a reward pool's reward-asset amount to the corresponding reward deposit-receipt token amount. This function uses `convertToReceiptTokenAmount` from `RewardsManagerCalculationsLib` to perform the conversion. If the undripped reward amount of the reward pool is zero, a floor value of one is used instead to prevent division by zero.

### Inputs

- `rewardPoolId_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the reward pool ID.
- `rewardAssetAmount_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Amount of the reward pool's asset to convert.

### Branches and code coverage

**Intended branches**

- Call `convertToReceiptTokenAmount` to convert amount and check division by zero.
  - ☑ Test coverage
- Return `depositReceiptTokenAmount_`, the corresponding amount of deposit-receipt tokens.
  - ☑ Test coverage

### Function: `convertRewardReceiptTokenToAssetAmount(uint16 rewardPoolId_, uint256 depositReceiptTokenAmount_)`

This function is used to convert a reward pool's reward deposit-receipt token amount to the corresponding reward-asset amount. This function uses `convertToAssetAmount` from `RewardsManagerCalculationsLib` to perform the conversion. If the undripped reward amount of the reward pool is zero, a floor value of one is used instead to prevent division by zero.

#### Inputs

- `rewardPoolId_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the reward-pool ID.
- `depositReceiptTokenAmount_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Amount of deposit-receipt tokens to convert.

#### Branches and code coverage

**Intended branches**

- Call `convertToAssetAmount` to convert amount and check division by zero.
  - ☑ Test coverage
- Return `rewardAssetAmount_`, the corresponding amount of the reward pool's asset.
  - ☑ Test coverage

### Function: `getClaimableRewards(uint16 stakePoolId_)`

This function is used to return all claimable rewards for a given stake pool.

#### Inputs

- `stakePoolId_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the stake-pool ID.

#### Branches and code coverage

**Intended branches**

- Return all claimable rewards for a given stake pool.
  - ☑ Test coverage

### Function: `getClaimableRewards()`

This function is used to return all claimable rewards for all stake pools and reward pools.

### Branches and code coverage

**Intended branches**

- Return all claimable rewards for all stake pools and reward pools.
  - ☑ Test coverage

## 5.7. Module: RewardsManager.sol

### Function: `initialize(address owner_, address pauser_, StakePoolConfig[] stakePoolConfigs_, RewardPoolConfig[] rewardPoolConfigs_)`

This function is used to initialize the RewardsManager with the provided parameters.

### Inputs

- `owner_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the owner of the RewardsManager.
- `pauser_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the pauser of the RewardsManager.
- `stakePoolConfigs_`
  - **Control**: Arbitrary.
  - **Constraints**: Validated in `isValidConfiguration`.
  - **Impact**: Array of stake pool configs.
- `rewardPoolConfigs_`
  - **Control**: Arbitrary.
  - **Constraints**: Validated in `isValidConfiguration`.
  - **Impact**: Array of reward pool configs.

### Branches and code coverage

**Intended branches**

- Initialize the RewardsManager accordingly.
  - [x] Test coverage

**Negative behavior**

- Revert if the RewardsManager is already initialized.
  - [x] Negative test
- Revert if the configuration is invalid.
  - [x] Negative test

## 5.8. Module: Staker.sol

### Function: `stakeWithoutTransfer(uint16 stakePoolId_, uint256 assetAmount_, address receiver_)`

This function is used to stake by minting `assetAmount_` stkReceiptTokens to `receiver_`. But this function expects that `assetAmount_` of `stakePoolId_` stake-pool asset has already been transferred to this RewardsManager contract.

### Inputs

- `stakePoolId_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the stake-pool ID to stake in.
- `assetAmount_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Amount of the underlying asset to stake.
- `receiver_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the receiver of the stkReceiptTokens.

### Branches and code coverage

**Intended branches**

- Checks contract has enough balance to stake.
  - [x] Test coverage

- Calls `_executeStake` to increase pool's state amount in order to update reward.
  - ☑ Test coverage
- Mints the stake-receipt token to receiver in `_executeStake`.
  - ☑ Test coverage

**Negative behavior**

- Revert if the amount is zero.
  - ☑ Negative test
- Revert if the amount that gets transferred is less than the amount that gets staked.
  - ☑ Negative test

### Function: `stake(uint16 stakePoolId_, uint256 assetAmount_, address receiver_, address from_)`

This function is used to stake by minting `assetAmount_` stkReceiptTokens to `receiver_` after depositing exactly `assetAmount_` of `stakePoolId_` stake-pool asset.

#### Inputs

- `stakePoolId_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the stake-pool ID to stake in.
- `assetAmount_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Amount of the underlying asset to stake.
- `receiver_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the receiver of the stkReceiptTokens.
- `from_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the sender of the underlying stake asset to the RewardsManager.

#### Branches and code coverage

**Intended branches**

- Calls `safeTransferFrom` to receive token.

- ☑ Test coverage
- Checks contract has enough balance to stake.
    - ☑ Test coverage
- Calls `_executeStake` to increase pool's state amount in order to update reward.
    - ☑ Test coverage
- Mints the stake-receipt token to receiver in `_executeStake`.
    - ☑ Test coverage

**Negative behavior**

- Revert if the amount is zero.
    - ☑ Negative test
- Revert if the amount that gets transferred is less than the amount that gets staked.
    - ☑ Negative test

### Function: `unstake(uint16 stakePoolId_, uint256 stkReceiptTokenAmount_, address receiver_, address owner_)`

This function is used to unstake by burning `stkReceiptTokenAmount_` of `stakePoolId_` stake-pool stake-receipt tokens and sending `stkReceiptTokenAmount_` of `stakePoolId_` stake-pool asset to `receiver_`. Also, it claims *all* outstanding user rewards and sends them to `receiver_`.

## Inputs

- `stakePoolId_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Value of the stake-pool ID to unstake from.
- `stkReceiptTokenAmount_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Amount of stkReceiptTokens to unstake.
- `receiver_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Address of the receiver of the unstaked assets.
- `owner_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Address of the owner of the stkReceiptTokens being unstaked.

### Branches and code coverage

**Intended branches**

- Calls `_claimRewards` to update and send reward to receiver.
  - ☑ Test coverage
- Subtracts amount from pool's state amount.
  - ☑ Test coverage
- Burns the stake-receipt token.
  - ☑ Test coverage
- Transfers the stake token to receiver.
  - ☑ Test coverage

**Negative behavior**

- Revert if the amount is zero.
  - ☑ Negative test

## 5.9. Module: StateChanger.sol

### Function: `pause()`

This function is used to pause the Rewards Manager. Only the owner, pauser, or Cozy manager can pause the Rewards Manager.

### Branches and code coverage

**Intended branches**

- Drip rewards.
  - ☑ Test coverage
- Pause the Rewards Manager.
  - ☑ Test coverage

**Negative behavior**

- Revert if the caller is not the owner, pauser, or Cozy manager.
  - ☑ Negative test
- Revert if the Rewards Manager is already paused.
  - ☑ Negative test

### Function: `unpause()`

This function is used to unpause the Rewards Manager. Only the owner or Cozy manager can unpause the Rewards Manager.

## Branches and code coverage

**Intended branches**

- Unpause the Rewards Manager.
    - [x] Test coverage
- Drip rewards.
    - [x] Test coverage

**Negative behavior**

- Revert if the caller is not the owner or Cozy manager.
    - [x] Negative test
- Revert if the Rewards Manager is already active.
    - [x] Negative test

# 5.10. Module: StkReceiptToken.sol

## Function: `transferFrom(address from_, address to_, uint256 amount_)`

This function is used to transfer the stkReceiptTokens from the `from_` address to the `to_` address. It updates the user's rewards before transferring the stkReceiptTokens by calling into the Rewards Manager.

## Inputs

- `from_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Address of the sender.
- `to_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Address of the recipient.
- `amount_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Amount of stkReceiptTokens to transfer.

## Branches and code coverage

**Intended branches**

- Invoke the `updateUserRewardsForStkReceiptTokenTransfer` function.

- [x] Test coverage

### Function: `transfer(address to_, uint256 amount_)`

This function is used to transfer the stkReceiptTokens from the caller to the recipient. It updates the user's rewards before transferring the stkReceiptTokens by calling into the Rewards Manager.

### Inputs

- `to_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Address of the recipient.
- `amount_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Amount of stkReceiptTokens to transfer.

### Branches and code coverage

**Intended branches**

- Invoke the `updateUserRewardsForStkReceiptTokenTransfer` function.
    - [x] Test coverage

## 5.11. Module: Configurator.sol

### Function: `finalizeUpdateConfigs(UpdateConfigsCalldataParams configUpdates_)`

Applies the queued update of configurations.

### Inputs

- `configUpdates_`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Its hash must match with the queued hash.
    - **Impact**: The given configuration is applied.

## Branches and code coverage

#### Intended branches

- Apply the configuration updates.
  - ☑ Test coverage
- Reset the queued configuration.
  - ☑ Test coverage

#### Negative behavior

- Revert if the Safety Module is in `TRIGGERED` state.
  - ☑ Negative test
- Revert if the delay for configuration update has not passed.
  - ☑ Negative test
- Revert if the deadline for configuration update has passed.
  - ☑ Negative test
- Revert if the hash of configuration does not match with the queued hash.
  - ☑ Negative test

## Function call analysis

- `ConfiguratorLib.finalizeUpdateConfigs(this.lastConfigUpdate, this.safetyModuleState, this.reservePools, this.triggerData, this.delays, this.receiptTokenFactory, configUpdates_)`
  - **What is controllable?** `configUpdates_`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Discarded.
  - **What happens if it reverts, reenters or does other unusual control flow?** Cannot reenter since the queued hash is changed to zero before any change.

### Function: `updateConfigs(UpdateConfigsCalldataParams configUpdates_)`

Queues the update of configurations.

## Inputs

- `configUpdates_`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be a valid configuration update.
  - **Impact**: Its hash and timestamp for update is stored in the contract.

### Branches and code coverage

**Intended branches**

- Update the configuration hash and timestamps accordingly.
  - ☑ Test coverage

**Negative behavior**

- Revert if the caller is not the Safety Module owner.
  - ☑ Negative test
- Revert if the given configuration is invalid.
  - ☑ Negative test

## 5.12. Module: CozySafetyModuleManager.sol

### Function: `claimFees(ISafetyModule[] safetyModules_)`

Claims fees from the Safety Modules in the given list.

### Inputs

- `safetyModules_`
  - **Control**: Fully controlled by any caller.
  - **Constraints**: None.
  - **Impact**: Safety Module contracts that dripped fees are claimed from.

### Branches and code coverage

**Intended branches**

- For each Safety Module, claims fees from the Safety Module to the protocol owner.
  - ☑ Test coverage

### Function call analysis

- `safetyModules_[i].claimFees(this.owner)`
  - **What is controllable?** Nothing.
  - **If the return value is controllable, how is it used and how can it go wrong?** Discarded.
  - **What happens if it reverts, reenters or does other unusual control flow?** Caller can exclude the reverting Safety Module from the list.

#### Function: `createSafetyModule(address owner_, address pauser_, UpdateConfigsCalldataParams configs_, byte[32] salt_)`

Deploys a new Safety Module.

### Inputs

- `owner_`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None.
    - **Impact**: Owner of the new Safety Module to be deployed.
- `pauser_`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None.
    - **Impact**: Pauser of the new Safety Module to be deployed.
- `configs_`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Must be a valid configuration for Safety Module.
    - **Impact**: Configuration of the new Safety Module to be deployed.
- `salt_`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None.
    - **Impact**: Solely affects the deterministically generated address of the Safety Module.

### Branches and code coverage

**Intended branches**

- The given configuration is correct, and the new Safety Module is deployed.
    - [x] Test coverage

**Negative behavior**

- Revert if the given configuration is invalid.
    - [x] Negative test
- The deployed address must be unique to the caller of the contract.
    - [ ] Negative test

### Function call analysis

- `ConfiguratorLib.isValidConfiguration(configs_.reservePoolConfigs, configs_.delaysConfig, this.allowedReservePools)`

  - **What is controllable?** `configs_.reservePoolConfigs` and `configs_.delaysConfig`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Safety Module with invalid configuration may be deployed.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `safetyModuleFactory_.computeAddress(salt_)`
  - **What is controllable?** `salt_`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Invalid address may be marked as the address of Safety Module.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `safetyModuleFactory_.deploySafetyModule(owner_, pauser_, configs_, salt_)`
  - **What is controllable?** All arguments.
  - **If the return value is controllable, how is it used and how can it go wrong?** The return value of this contract can be manipulated.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.

### Function: `pause(ISafetyModule[] safetyModules_)`

Pauses the Safety Modules in the given list.

### Inputs

- `safetyModules_`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Listed Safety Modules are paused.

### Branches and code coverage

**Intended branches**

- Pause the Safety Modules in the list.
  - ☑ Test coverage

**Negative behavior**

- Revert if the caller is not the protocol owner or pauser.
  - ☑ Negative test

### Function call analysis

- `safetyModules_[i].pause()`
  - **What is controllable?** Nothing.

- **If the return value is controllable, how is it used and how can it go wrong?** Discarded.
- **What happens if it reverts, reenters or does other unusual control flow?** Caller can exclude the reverting Safety Module from the list.

### Function: `unpause(ISafetyModule[] safetyModules_)`

Unpauses the Safety Modules in the given list.

#### Inputs

- `safetyModules_`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: Listed Safety Modules are paused.

#### Branches and code coverage

**Intended branches**

- Unpause the Safety Modules in the list.
  - ☑ Test coverage

**Negative behavior**

- Revert if the caller is not the protocol owner.
  - ☑ Negative test
- Revert if the caller is the protocol pauser.
  - ☑ Negative test

#### Function call analysis

- `safetyModules_[i].unpause()`
  - **What is controllable?** Nothing.
  - **If the return value is controllable, how is it used and how can it go wrong?** Discarded.
  - **What happens if it reverts, reenters or does other unusual control flow?** Caller can exclude the reverting Safety Module from the list.

## 5.13. Module: Depositor.sol

### Function: `depositReserveAssets(uint8 reservePoolId_, uint256 reserveAssetAmount_, address receiver_, address from_)`

Transfers the asset from `from_` to this contract and processes the deposit for `receiver`.

### Inputs

- `reservePoolId_`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: The asset of the reserve pool will be transferred and deposited.
- `reserveAssetAmount_`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Amount to be deposited.
- `receiver_`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Receiver of deposit-receipt token.
- `from_`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Address that the asset to be deposited is transferred from.

### Branches and code coverage

**Intended branches**

- Asset is transferred from `from_`.
  - [x] Test coverage
- Deposit is processed by calling `_executeReserveDeposit`.
  - [x] Test coverage

**Negative behavior**

- The caller is `from_` or is allowed to use the allowances of `from_`.
  - [ ] Negative test

## 5.14. Module: FeesHandler.sol

### Function: `claimFees(address owner_)`

Claims any accrued fees.

#### Inputs

- `owner_`
    - **Control**: Owner of Safety Module Manager.
    - **Constraints**: None.
    - **Impact**: Accrued fees are transferred to this address.

#### Branches and code coverage

**Intended branches**

- Fetch the fee-dripping model from the Safety Module Manager.
    - ☑ Test coverage
- Send the accrued fees to the `owner_`.
    - ☑ Test coverage

**Negative behavior**

- Reverts if the `msg.sender` is not the Safety Module Manager.
    - ☑ Negative test

### Function: `_dripFeesFromReservePool(ReservePool reservePool_, IDripModel dripModel_)`

Drips fees from the specified reserve pool.

#### Inputs

- `reservePool_`
    - **Control**: Valid reserve pool registered in the Safety Module.
    - **Constraints**: None.
    - **Impact**: Fees are dripped from this reserve pool.
- `dripModel_`
    - **Control**: Owner of Safety Module Manager.
    - **Constraints**: None.
    - **Impact**: The fees to be dripped are calculated here.

## Branches and code coverage

**Intended branches**

- Calculate the fees to be dripped.
  - ☑ Test coverage
- The amount of fees are moved from `depositAmount` to `feeAmount` of the reserve pool.
  - ☑ Test coverage
- Update the last time of fee dripping.
  - ☑ Test coverage

**Negative behavior**

- Do not reenter before changing the last time of fee dripping.
  - ☑ Negative test

## 5.15. Module: Redeemer.sol

### Function: `completeRedemption(uint64 redemptionId_)`

Completes the redemption request for the specified redemption ID.

## Inputs

- `redemptionId_`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: An ID of the redemption to be completed.

## Branches and code coverage

**Intended branches**

- Remove the redemption from queue.
  - ☑ Test coverage
- Transfer the asset to the expected receiver.
  - ☑ Test coverage

**Negative behavior**

- Reverts if the delay has not passed.
  - ☑ Negative test
- Reverts if the Safety Module is triggered.
  - ☑ Negative test

### Function: `redeem(uint8 reservePoolId_, uint256 depositReceiptTokenAmount_, address receiver_, address owner_)`

Queues a redemption by burning deposit-receipt tokens.

#### Inputs

- `reservePoolId_`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: The ID of the reserve pool that the assets are to be redeemed.
- `depositReceiptTokenAmount_`
  - **Control**: Full.
  - **Constraints**: Must be lower than or equal to the deposit-receipt token balance of `owner_`.
  - **Impact**: The amount of deposit-receipt token to be redeemed.
- `receiver_`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: A user who receives the corresponding assets.
- `owner_`
  - **Control**: Full.
  - **Constraints**: Must be `msg.sender` or approve the `msg.sender` at least `depositReceiptTokenAmount_`.
  - **Impact**: A user whose deposit-receipt tokens are burnt.

#### Branches and code coverage

**Intended branches**

- Drip fees before processing the redemption.
  - ☑ Test coverage
- Queue a new redemption.
  - ☑ Test coverage

**Negative behavior**

- Revert if the Safety Module is triggered.
  - ☑ Negative test
- Revert if no asset can be redeemed back to a user.
  - ☑ Negative test

## 5.16. Module: SafetyModuleFactory.sol

### Function: `deploySafetyModule(address owner_, address pauser_, UpdateConfigsCalldataParams configs_, byte[32] baseSalt_)`

Deploy a Safety Module.

### Inputs

- `owner_`
  - **Control**: Fully controlled by the caller who called the Safety Module Manager.
  - **Constraints**: None.
  - **Impact**: Owner of the new Safety Module to be deployed.
- `pauser_`
  - **Control**: Fully controlled by the caller who called the Safety Module Manager.
  - **Constraints**: None.
  - **Impact**: Pauser of the new Safety Module to be deployed.
- `configs_`
  - **Control**: Fully controlled by the caller who called the Safety Module Manager.
  - **Constraints**: Must be a valid configuration for Safety Module.
  - **Impact**: Configuration of the new Safety Module to be deployed.
- `baseSalt_`
  - **Control**: Fully controlled by the caller who called the Safety Module Manager.
  - **Constraints**: None.
  - **Impact**: Solely affects the deterministically generated address of the Safety Module.

### Branches and code coverage

**Intended branches**

- Deploy and initialize a Safety Module.
  - ☑ Test coverage

**Negative behavior**

- Revert if the caller is not the Safety Module Manager.
  - ☑ Negative test

### Function call analysis

- `Clones.cloneDeterministic(address(this.safetyModuleLogic), this.salt(baseSalt_))`

   - **What is controllable?** `baseSalt_`.
   - **If the return value is controllable, how is it used and how can it go wrong?** The returned Safety Module address may be wrong.
   - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `safetyModule_.initialize(owner_, pauser_, configs_)`
   - **What is controllable?** All arguments.
   - **If the return value is controllable, how is it used and how can it go wrong?** Discarded.
   - **What happens if it reverts, reenters or does other unusual control flow?** N/A.

## 5.17. Module: SlashHandler.sol

### Function: `slash(Slash[] slashes_, address receiver_)`

Slashes the reserve pools.

#### Inputs

- `slashes_`
   - **Control**: Payout handler controls list of slashes.
   - **Constraints**: Slashed reserves must be unique. Amounts must be lower than the maximum slash amount.
   - **Impact**: Reserves and amounts to be slashed.
- `receiver_`
   - **Control**: Payout handler controls receiver of the slashed assets.
   - **Constraints**: None.
   - **Impact**: Account that receives the slashed assets.

#### Branches and code coverage

**Intended branches**

- Subtract the slash amount from the reserve/asset pool.
   - ☑ Test coverage
- Transfer the slashed assets to the receiver.
   - ☑ Test coverage
- Change the state of Safety Module correctly.
   - ☑ Test coverage

**Negative behavior**

- Revert if a payout handler has no pending trigger.
   - ☑ Negative test

- Revert if a reserve pool is slashed multiple times.
  - ☑ Negative test
- Reverts if slash amount exceeds the maximum slash amount.
  - ☑ Negative test

### Function: `_updateAlreadySlashed(uint256 alreadySlashed_, uint8 poolId_)`

Updates the bitmap used to track which reserve pools have already been slashed.

#### Inputs

- `alreadySlashed_`
  - **Control**: Managed by `slash` function.
  - **Constraints**: Must be the previous return value of the `_updateAlreadySlashed` function.
  - **Impact**: Bitmap marked with the slashed pools.
- `poolId_`
  - **Control**: Full through `slash` function.
  - **Constraints**: The pool must not be recorded in the current bitmap.
  - **Impact**: Marked in the returned bitmap.

#### Branches and code coverage

**Intended branches**

- Add the pool to the bitmap.
  - ☑ Test coverage

**Negative behavior**

- Reverts if the pool is already marked in the bitmap.
  - ☑ Negative test

## 5.18. Module: StateChanger.sol

### Function: `pause()`

Pauses the Safety Module.

#### Branches and code coverage

**Intended branches**

- Drip fees from the Safety Module.
  - ☑ Test coverage
- Change the state to `PAUSED`.
  - ☑ Test coverage

**Negative behavior**

- Revert if the caller does not have permission for pausing the Safety Module.
  - ☑ Negative test
- Revert if the state transition is not valid.
  - ☑ Negative test

### Function call analysis

- `this.dripFees() -> this._dripFeesFromReservePool(this.reservePools[i], dripModel_) -> this._getNextDripAmount(reservePool_.depositAmount - reservePool_.pendingWithdrawalsAmount, dripModel_, reservePool_.lastFeesDripTime) -> dripModel_.dripFactor(lastDripTime_)`
  - **What is controllable?** None.
  - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reentering pause does not have side effects.

## Function: `trigger(ITrigger trigger_)`

Triggers the Safety Module.

### Inputs

- `trigger_`
  - **Control**: Full.
  - **Constraints**: Must be valid trigger registered in Safety Module.
  - **Impact**: Contract with slashing-condition logic.

### Branches and code coverage

**Intended branches**

- Drip fees from the Safety Module.
  - ☑ Test coverage
- Change the state to `TRIGGERED`.
  - ☑ Test coverage

- Increment the counters for slashing.
  - ☑ Test coverage

**Negative behavior**

- Revert if the trigger is not in `TRIGGERED` state.
  - ☑ Negative test

## Function call analysis

- `trigger_.state()`
  - **What is controllable?** None.
  - **If the return value is controllable, how is it used and how can it go wrong?** The Safety Module can be triggered without satisfying condition.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this.dripFees() -> this.cozySafetyModuleManager.getFeeDripModel(ISafetyModule(ad`
  - **What is controllable?** None.
  - **If the return value is controllable, how is it used and how can it go wrong?** Fee dripping model can be changed by Safety Module Manager.
  - **What happens if it reverts, reenters or does other unusual control flow?** Cannot reenter as there is no untrusted external interactions.
- `this.dripFees() -> this._dripFeesFromReservePool(this.reservePools[i], dripModel_) -> this._getNextDripAmount(reservePool_.depositAmount - reservePool_.pendingWithdrawalsAmount, dripModel_, reservePool_.lastFeesDripTime) -> dripModel_.dripFactor(lastDripTime_)`
  - **What is controllable?** None.
  - **If the return value is controllable, how is it used and how can it go wrong?** Safety Module.
  - **What happens if it reverts, reenters or does other unusual control flow?** Cannot reenter.

## Function: `unpause()`

Unpauses the Safety Module.

## Branches and code coverage

**Intended branches**

- Change the state to `ACTIVE` or `TRIGGERED`.
  - ☑ Test coverage
- Drip fees from the Safety Module.
  - ☑ Test coverage

**Negative behavior**

- Revert if the caller does not have permission for pausing the Safety Module.
  - ☑ Negative test
- Revert if the state transition is not valid.
  - ☑ Negative test
- Do not drip fees if the new state is `TRIGGERED`.
  - ☑ Negative test

### Function call analysis

- `this.dripFees() -> this._dripFeesFromReservePool(this.reservePools[i], dripModel_) -> this._getNextDripAmount(reservePool_.depositAmount - reservePool_.pendingWithdrawalsAmount, dripModel_, reservePool_.lastFeesDripTime) -> dripModel_.dripFactor(lastDripTime_)`
  - **What is controllable?** None.
  - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reentering unpause does not have side effects.

## 5.19. Module: StateTransitionsLib.sol

### Function: `isValidStateTransition(CallerRole role_, SafetyModuleState to_, SafetyModuleState from_, bool nonZeroPendingSlashes_)`

This function checks if the state transition is valid. It returns true if the state transition is valid, false otherwise. This check is based on the role of the caller, the state being transitioned to, the state being transitioned from, and the number of pending slashes.

### Inputs

- `role_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the role of the caller.
- `to_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the state that is being transitioned to.
- `from_`
  - **Control**: Arbitrary.
  - **Constraints**: None.

    - **Impact**: Value of the state that is being transitioned from.
- `nonZeroPendingSlashes_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Value of the number of pending slashes.

### Branches and code coverage

**Intended branches**

- Return true if the state transition is valid.
    - ☑ Test coverage
- Return false if the state transition is not valid.
    - ☑ Test coverage

## 5.20. Module: shared ReceiptTokenFactory.sol

### Function: `computeAddress(address module_, uint16 poolId_, PoolType poolType_)`

This function is used to return the address of the ReceiptToken, which is deployed by the module with the given pool ID and pool type.

### Inputs

- `module_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Address of module.
- `poolId_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Value of pool ID.
- `poolType_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Value of pool type.

### Branches and code coverage

**Intended branches**

- Return computed contract address using `predictDeterministicAddress`.
    - ☑ Test coverage

### Function: `deployReceiptToken(uint16 poolId_, PoolType poolType_, uint8 decimals_)`

This function is used to deploy a new ReceiptToken contract with the given pool ID, pool type, and decimals.

#### Inputs

- `poolId_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Value of pool ID.
- `poolType_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Value of pool type.
- `decimals_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Value of decimals.

#### Branches and code coverage

**Intended branches**

- Deploy new ReceiptToken using `cloneDeterministic`.
    - ☑ Test coverage
- Call initialize of ReceiptToken.
    - ☑ Test coverage

### Function: `salt(address module_, uint16 poolId_, PoolType poolType_)`

This function is used to return the salt that is used to compute the ReceiptToken address.

#### Inputs

- `module_`
    - **Control**: Arbitrary.

    - **Constraints**: None.
    - **Impact**: Address of module.
- `poolId_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Value of pool ID.
- `poolType_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Value of pool type.

### Branches and code coverage

#### Intended branches

- Return `keccak256` hash of parameters.
    - ☑ Test coverage

## 5.21. Module: shared ReceiptToken.sol

### Function: `burn(address caller_, address owner_, uint256 amount_)`

This function is used to burn the token from the owner.

### Inputs

- `caller_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Address of the caller of burn.
- `owner_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Address of the owner of the token to be burned.
- `amount_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Amount of token to be burned.

## Branches and code coverage

**Intended branches**

- Burn the token.
  - ☑ Test coverage

**Negative behavior**

- Revert if caller is not the module.
  - ☑ Negative test

## Function: `initialize(address module_, string name_, string symbol_, uint8 decimals_)`

This function is used to initialize the token.

## Inputs

- `module_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the module.
- `name_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: String of the name of the token.
- `symbol_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: String of the symbol of the token.
- `decimals_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the decimal of the token.

## Branches and code coverage

**Intended branches**

- Initialize token with module address.
  - ☑ Test coverage

**Negative behavior**

- Revert if the module is already initialized.
    - ☑ Negative test

## Function: `mint(address to_, uint256 amount_)`

This function is used to mint the token to the receiver.

### Inputs

- `to_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Address of the receiver of minted token.
- `amount_`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Amount of token to be minted.

### Branches and code coverage

**Intended branches**

- Mint the token.
    - ☑ Test coverage

**Negative behavior**

- Revert if caller is not the module.
    - ☑ Negative test

# 6. Assessment Results

At the time of our assessment, the reviewed code was not deployed to the Ethereum Mainnet.

During our assessment on the scoped Safety Module contracts, we discovered eight findings. No critical issues were found. Two findings were of high impact, four were of medium impact, and two were of low impact. Cozy Finance Inc. acknowledged all findings and implemented fixes.

## 6.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.