SOLIDIFIED

Audit Report for HAI - November 24, 2023

Summary

Audit Report prepared by Solidified covering HAI's framework for creating stablecoins.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on October 5, 2023, and the results are presented here.

Audited Files

The source code has been supplied in the following source code repository:

Repo: https://github.com/hai-on-op/core
Commit hash: `2b9cf1415b5dee6aee5511f151d2da6765052a31`

```
src
├── contracts
│    ├── AccountingEngine.sol
│    ├── CollateralAuctionHouse.sol
│    ├── DebtAuctionHouse.sol
│    ├── LiquidationEngine.sol
│    ├── OracleRelayer.sol
│    ├── PIDController.sol
│    ├── PIDRateSetter.sol
│    ├── SAFEEngine.sol
│    ├── StabilityFeeTreasury.sol
│    ├── SurplusAuctionHouse.sol
│    ├── TaxCollector.sol
│    ├── factories
│    │    ├── AuthorizableChild.sol
│    │    ├── ChainlinkRelayerChild.sol
│    │    ├── ChainlinkRelayerFactory.sol
│    │    ├── CollateralAuctionHouseChild.sol
│    │    ├── CollateralAuctionHouseFactory.sol
│    │    ├── CollateralJoinChild.sol
│    │    ├── CollateralJoinDelegatableChild.sol
│    │    ├── CollateralJoinFactory.sol
│   │   ├── DelayedOracleChild.sol
│   │   ├── DelayedOracleFactory.sol
│   │   ├── DenominatedOracleChild.sol
│   │   ├── DenominatedOracleFactory.sol
│   │   ├── DisableableChild.sol
│   │   ├── FactoryChild.sol
│   │   ├── UniV3RelayerChild.sol
│   │   └── UniV3RelayerFactory.sol
│   ├── for-test
│   │   ├── DeviatedOracle.sol
│   │   ├── HardcodedOracle.sol
│   │   └── MintableERC20.sol
│   ├── jobs
│   │   ├── AccountingJob.sol
│   │   ├── Job.sol
│   │   ├── LiquidationJob.sol
│   │   └── OracleJob.sol
│   ├── oracles
│   │   ├── ChainlinkRelayer.sol
│   │   ├── DelayedOracle.sol
│   │   ├── DenominatedOracle.sol
│   │   └── UniV3Relayer.sol
│   ├── proxies
│   │   ├── HaiProxy.sol
│   │   ├── HaiProxyFactory.sol
│   │   ├── HaiProxyRegistry.sol
│   │   ├── HaiSafeManager.sol
│   │   ├── SAFEHandler.sol
│   │   └── actions
│   │       ├── BasicActions.sol
│   │       ├── CollateralBidActions.sol
│   │       ├── CommonActions.sol
│   │       ├── DebtBidActions.sol
│   │       ├── GlobalSettlementActions.sol
│   │       ├── PostSettlementSurplusBidActions.sol
│   │       ├── RewardedActions.sol
│   │       └── SurplusBidActions.sol
│   ├── settlement
│   │   ├── GlobalSettlement.sol
│   │   ├── PostSettlementSurplusAuctionHouse.sol
│   │   └── SettlementSurplusAuctioneer.sol
│   ├── tokens
│   │   ├── ProtocolToken.sol
│   │   ├── SystemCoin.sol
│   │   └── TokenDistributor.sol
│   └── utils
│       ├── Authorizable.sol
│       ├── CoinJoin.sol
│       ├── CollateralJoin.sol
│       ├── Disableable.sol
│       ├── ETHJoin.sol
│       ├── Modifiable.sol
│       └── Ownable.sol
├── interfaces
│   ├── IAccountingEngine.sol
│   ├── ICollateralAuctionHouse.sol
│   ├── ICommonSurplusAuctionHouse.sol
│   ├── IDebtAuctionHouse.sol
│   ├── ILiquidationEngine.sol
│   ├── IOracleRelayer.sol
│   ├── IPIDController.sol
│   ├── IPIDRateSetter.sol
│   ├── ISAFEEngine.sol
│   ├── IStabilityFeeTreasury.sol
│   ├── ISurplusAuctionHouse.sol
│   ├── ITaxCollector.sol
│   ├── external
│   │   ├── ISAFESaviour.sol
│   │   └── IWeth.sol
│   ├── factories
│   │   ├── IAuthorizableChild.sol
│   │   ├── IChainlinkRelayerChild.sol
│   │   ├── IChainlinkRelayerFactory.sol
│   │   ├── ICollateralAuctionHouseChild.sol
│   │   ├── ICollateralAuctionHouseFactory.sol
│   │   ├── ICollateralJoinChild.sol
│   │   ├── ICollateralJoinDelegatableChild.sol
│   │   ├── ICollateralJoinFactory.sol
│   │   ├── IDelayedOracleChild.sol
│   │   ├── IDelayedOracleFactory.sol
│   │   ├── IDenominatedOracleChild.sol
│   │   ├── IDenominatedOracleFactory.sol
│   │   ├── IDisableableChild.sol
│   │   ├── IFactoryChild.sol
│   │   ├── IUniV3RelayerChild.sol
│   │   └── IUniV3RelayerFactory.sol
│   ├── jobs
│   │   ├── IAccountingJob.sol
│   │   ├── IJob.sol
│   │   ├── ILiquidationJob.sol
│   │   └── IOracleJob.sol
│   ├── oracles
│   │   ├── IBaseOracle.sol
│   │   ├── IChainlinkOracle.sol
│   │   ├── IChainlinkRelayer.sol
│   │   ├── IDelayedOracle.sol
│   │   ├── IDenominatedOracle.sol
│   │   └── IUniV3Relayer.sol
│   ├── proxies
│   │   ├── IHaiProxy.sol
│   │   ├── IHaiProxyFactory.sol
│   │   ├── IHaiProxyRegistry.sol
│   │   ├── IHaiSafeManager.sol
│   │   └── actions
│   │       ├── IBasicActions.sol
│   │       ├── ICollateralBidActions.sol
│   │       ├── ICommonActions.sol
│   │       ├── IDebtBidActions.sol
│   │       ├── IGlobalSettlementActions.sol
│   │       ├── IRewardedActions.sol
│   │       └── ISurplusBidActions.sol
│   ├── settlement
│   │   ├── IGlobalSettlement.sol
│   │   ├── IPostSettlementSurplusAuctionHouse.sol
│   │   └── ISettlementSurplusAuctioneer.sol
│   ├── tokens
│   │   ├── IProtocolToken.sol
│   │   ├── ISystemCoin.sol
│   │   └── ITokenDistributor.sol
│   └── utils
│       ├── IAuthorizable.sol
│       ├── ICoinJoin.sol
│       ├── ICollateralJoin.sol
│       ├── IDisableable.sol
│       ├── IETHJoin.sol
│       ├── IModifiable.sol
│       └── IOwnable.sol
└── libraries
    ├── Assertions.sol
    ├── Encoding.sol
    └── Math.sol
```

Intended Behavior

HAI, a fork of RAI, is a framework for creating stablecoins that are pegged to the US dollar. A diverse basket of collateral types backs the minting of the stablecoin.

Findings

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	Status	Comment
Code complexity	Medium	The protocol contains many different contracts and modules, each one responsible for a part of HAI. The separation of concerns makes it easy to analyze each component individually, and the minor differences from GEB allow users to leverage the existing documentation from the forked protocol to quickly understand the new system.
Code readability and clarity	High	The protocol codebase is well-structured and easy to read and maintain.
Level of Documentation	High	The documentation is extensive, both with regard to the NatSpec comments and to the high-level overview documentation website.
Test Coverage	High	The test coverage is extensive and contains many different

		scenarios and functionalities.

Issues Found

Solidified found that the HAI contracts contain no critical issues, 3 major issues, 12 minor issues, and 12 informational notes.

We recommend issues are amended, while informational notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	Anyone can disable the protection of a SAFE	Major	Resolved
2	Missing debt check lets users start a debt auction of non-existent debt	Major	Resolved
3	Liquidation DOS	Major	Resolved
4	UniV3Relayer does not handle tokens with more than 18 decimals	Minor	Acknowledged
5	DenominatedOracle.getResultWithValidity may revert	Minor	Resolved
6	TokenDistributor.withdraw can be used to sweep tokens before the claim period has ended	Minor	Resolved
7	ChainlinkRelayer.read handles negative values wrong	Minor	Resolved
8	SurplusAuctionHouse refunds initial bid	Minor	Resolved
9	TokenDistributor.claimAndDelegate uses hardcoded nonce 0	Minor	Resolved
10	HaiSafeManager.transferSAFEOwnership uses wrong SAFE ID	Minor	Resolved
11	Missing sequencer uptime check for L2 ChainlinkRelayer	Minor	Resolved
12	Centralization risk	Minor	Acknowledged

13	HaiSafeManager does not clean up savior information when users abandon their SAFEs	Minor	Resolved
14	Missing input validation in some contracts extending from Modifiable	Minor	Resolved
15	Precision loss on discountedPrice calculation	Minor	Resolved
16	No backup oracle implemented	Note	Acknowledged
17	Contracts should implement a two-step ownership transfer	Note	Resolved
18	HaiSafeManager.openSAFE can be used to open SAFEs for other users and HaiSafeManager.addSAFE can be used to add SAFEs from other users	Note	Acknowledged
19	Mismatch between documentation and implementation	Note	Resolved
20	Typos	Note	Resolved
21	Unused return values	Note	Resolved
22	Incorrect parameter emitted on events	Note	Resolved
23	Missing revert string on HaiProxyRegistry function	Note	Acknowledged
24	Protocol does not support rebasing/fee-on-transfer tokens	Note	Acknowledged
25	Possible reentrancy if ERC777 tokens are used as collateral	Note	Resolved
26	Missing whenEnabled modifier	Note	Acknowledged
27	Unsafe cast to int256	Note	Acknowledged

Critical Issues

No critical issues have been found.

Major Issues

1. Anyone can disable the protection of a SAFE

The function `LiquidationEngine.protectSAFE` requires that the caller is allowed to modify a SAFE. However, this check is only performed when the argument `_saviour` is not equal to `address(0)`. But the zero address is also a valid argument for the function, as it disables the protection for a given collateral type and safe address. The missing authorization check therefore allows anyone to disable the protection of a SAFE. An attacker can use this vulnerability to call `liquidateSAFE` on `_safe` addresses that would be otherwise protected by their saviours by first removing them from `chosenSAFESaviour` and then performing the liquidation.

Recommendation

We recommend also performing an authorization check when the address is equal to the zero address.

2. Missing debt check lets users start a debt auction of non-existent debt

The `AccountingEngine.sol` contract serves as the protocol's central component responsible for initializing both debt and surplus auctions. Debt auctions are consistently initiated with a predefined minimum bid referred to as debtAuctionBidSize. This is done to ensure that the protocol can only auction debt that is not currently undergoing an auction and is not locked within the debt queue, as articulated in the comment found on `IAccountingEngine:246`: "It can only auction debt that has been popped from the debt queue and is not already being auctioned". This necessity prompts the check on `AccountingEngine:317`:

```
if (_params.debtAuctionBidSize > _unqueuedUnauctionedDebt(_debtBalance))
```

This check verifies that there is a sufficient amount of bad debt available to auction.

The issue stems in the line 180, where `_settleDebt` is called, this aims to ensure that only bad debt is considered for the auction. However, if the remaining bad debt, after settlement, falls below the specified threshold `(debtAuctionBidSize <= unqueuedUnauctionedDebt())`, the auction still starts with an incorrect amount of bad debt coverage, diluting the protocol Token when it is not yet needed.

Recommendation
To mitigate this risk, we suggest introducing the check `(_params.debtAuctionBidSize > _unqueuedUnauctionedDebt(_debtBalance))` after calling `_settleDebt` to ensure there exists enough amount of bad in the contract after the settling.

3. Liquidation DOS

In `LiquidationEngine` line 186, if a SAFE's debt amount is going to be left under `debtFloor` after the liquidation, the operation is reverted. A malicious user could monitor the system to keep his debt between that threshold, tricking the system into not being liquidated.

Note: This issue was initially rated a minor, but has been updated to major severity after some internal discussion

Recommendation
To resolve this matter, our suggestion is to fully liquidate the debt of a SAFE if the remaining debt falls below the debt floor, instead of reverting the transaction.

Minor Issues

4. `UniV3Relayer` does not handle tokens with more than 18 decimals

The variable `multiplier` in UniV3Relayer.sol is set to `18 - IERC20Metadata(_quoteToken).decimals()`. Because the variable is an unsigned integer, it will underflow and the deployment will fail for quote tokens with more than 18 decimals.

Recommendation
Consider supporting tokens with more than 18 decimals.

5. `DenominatedOracle.getResultWithValidity` may revert

The oracles are designed in a way such that `getResultWithValidity` does not revert for invalid values and states, but returns `false` instead. However, `DenominatedOracle` does not always adhere to this interface. When `denominationPriceSource` returns 0 as the price value, the oracle will try to perform a division by zero, causing a revert. This scenario can for instance occur in combination with `UniV3Relayer`, which returns zero when the pool does not have enough historical values.

Recommendation
Check for a value of 0 and set the validity flag to `false` in such a scenario.

6. `TokenDistributor.withdraw` can be used to sweep tokens before the claim period has ended

The `TokenDistributor` contract contains a `sweep` function, which can be used to retrieve all remaining tokens after the claim period has ended. However, it also has a function `withdraw` that allows an authorized user to withdraw tokens at any time. It is therefore not ensured that the contract will contain the necessary amount of tokens and a malicious authorized user could withdraw them.

Recommendation

Consider removing the `withdraw` function or limiting its functionality.

7. `ChainlinkRelayer.read` handles negative values wrong

The function `ChainlinkRelayer.getResultWithValidity` correctly checks if the result is greater than zero and sets the validity flag to `false` if not. However, the logic in the `read` function is different. The function only reverts when the value is zero. A negative value is cast to an unsigned integer, which results in completely wrong prices.

Recommendation

Revert when the price is negative.

8. `SurplusAuctionHouse` refunds initial bid

A user can specify an initial bid value when starting an auction via `SurplusAuctionHouse.startAuction`. The function sets the initial bid expiry value to 0. When someone tries to increase the bid via `increaseBidSize`, the function tries to send back the funds to the initial bidder (the creator of the auction), although they never sent funds for the creation.

Recommendation

Do not return funds when the initial bid is increased.

9. `TokenDistributor.claimAndDelegate` uses hardcoded nonce 0

The function TokenDistributor.claimAndDelegate always passes the nonce 0 to `delegateBySig`. Because this may not be the current nonce of the user, the functionality of the function is severely limited.

Recommendation
Allow the user to specify the nonce.

10. `HaiSafeManager.transferSAFEOwnership` uses wrong SAFE ID

The function `transferSAFEOwnership` uses the variables `_safeId` (which is an incremental nonce that is used when creating SAFEs) instead of `_safe` (which is the value that the user requested) when modifying the entries in `_usrSafesPerCollat`. Because this mapping is only used in a view function, the impact of the error is somewhat limited and it does not enable direct attacks.

Recommendation
Use the correct ID when modifying the entries.

11. Missing sequencer uptime check for L2 `ChainlinkRelayer`

In `src/contracts/oracles/ChainlinkRelayer.sol`, the `_isValidFeed` function does not perform an L2 sequencer uptime verification before returning the `_valid` boolean value. As a result, since HAI is intended to be deployed on Optimism, if the sequencer that executes and rolls up the L2 transactions is unavailable, continuing to serve price feed data can mean relying on stale information.

Recommendation

We recommend that Optimistic L2 oracles consult the Sequencer Uptime Feed to ensure that the sequencer is live before trusting the data returned by the oracle.

12. Centralization risk

The HAI protocol contains several functions guarded by the `isAuthorized` modifier, that are meant to be called by authorized accounts and contracts to perform important bookkeeping tasks. In the event of improper configuration or loss of administration private keys, the protocol funds can be stolen due to the manipulation of the project through several functions. For example, an attacker can mint or burn `SystemCoin`s to and from any address, start and terminate auctions, add SAFE saviors with potentially malicious code, remove working SAFE saviors that users rely on, change a SAFE's collateral and debt, update system parameters, among others.

Recommendation

We recommend ensuring that the authorized accounts and contracts are properly configured and making sure that any private keys are properly managed to avoid any potential risks of compromise. In addition, we recommend implementing multi-signature wallets whenever possible for an additional layer of security.

13. `HaiSafeManager` does not clean up savior information when users abandon their SAFEs

In `src/contracts/proxies/HaiSafeManager`, the SAFE's savior information is not cleaned up when the SAFE is abandoned by a user, either through `transferSAFEOwnership`, `moveSAFE`, `removeSAFE`, or `quitSystem`. This may not be expected or wanted by the users abandoning their SAFEs, especially since some of these actions have the connotation of completely erasing the previous SAFE information.

We classify this issue as minor since the user can always call `protectSAFE` before executing any of these functions if they wish to clean up the savior information.

Recommendation

We recommend cleaning up the SAFE savior information whenever the safe ID is removed from the list of SAFEs from a user. Alternatively, we recommend thoroughly documenting the current behavior of the `HaiSafeManager` in order to inform users that the SAFE savior is not changed when the user abandons their SAFEs.

14. Missing input validation in some contracts extending from `Modifiable`

In `src/contracts/jobs/AccountingJob.sol` and `src/contracts/jobs/LiquidationJob.sol`, and `src/contracts/jobs/OracleJob.sol` the contracts extend from `Modifiable` but they do not implement `_validateParameters`. As a result, if the administrator performs an update with invalid parameters, wrong information can be set into the contract's state variables.

Recommendation

We recommend implementing the `_validateParameters` function on these contracts.

15. Precision loss on `discountedPrice` calculation

In `CollateralAuctionHouse` line `122`, the `_discountedPrice` value is calculated by dividing `_collateralPrice` by `_systemCoinPrice` and multiplying the result by the `_customDiscount`. This order of operations leads to precision loss in favor of the user at expense of the protocol rendering the discounted price a bit lower than it should be.

Recommendation

To address this issue, we recommend switching the order of operations to the following:

```
uint256 _discountedPrice =
_collateralPrice.wmul(_customDiscount).rdiv(_systemCoinPrice);
```

Informational Notes

16. No backup oracle implemented

Within the `OracleRelayer` contract, a prerequisite for updating the collateral price is obtaining a valid price result from the oracle associated with the specific collateral type (cType). In the `ChainlinkRelayer` contract, a result might be deemed invalid under two conditions: firstly, if the round data becomes outdated; and secondly, as mentioned in the aforementioned issue, if the sequencer encounters a period of inactivity. In such instances, the protocol will use a price of 0 for all subsequent operations. While this is safe, some actions (such as increasing debt) will not be possible and the functionality will be restricted.

Recommendation
Consider implementing a backup oracle. For instance, a Uniswap TWAP oracle could be used as a backup mechanism.

17. Contracts should implement a two step ownership transfer

The contracts within the scope of this audit allow the current owner to execute a one-step ownership transfer. While this is common practice, it presents a risk for the ownership of the contract to become lost if the owner transfers ownership to the incorrect address. A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and actually execute the config update.

Recommendation
We recommend implementing a two-step ownership transfer.

18. `HaiSafeManager.openSAFE` can be used to open SAFEs for other users and `HaiSafeManager.addSAFE` can be used to add SAFEs from other users

The function `openSAFE` has a `_usr` argument which is used to specify the owner of the SAFE. This allows anyone to create SAFEs for other users. In addition, users can call `addSAFE` to add a third party's SAFE to their lists of `_usrSafes` and `_usrSafesPerCollat`, even though they are not the owners of this safe ID. Because the contract uses the `EnumerableSet` library where most operations are not linear in the size of the set, this cannot be directly abused for Denial of Service attacks on users. Nevertheless, this architecture can be confusing for external systems relying solely on the information returned by `getSafes`, which returns the list of safe IDs that may not represent the SAFEs created or owned by the user. Finally, the function `getSafes` calls `values()` on the `EnumerableSet`, which has a linear complexity and may therefore run out of gas when a user has too many SAFEs.

We classify this issue as informational since an attacker cannot exploit this fact to gain ownership of another user's SAFE or any of their assets. In addition, a user can always call `removeSAFE` to clean up any unwanted safe IDs from their list of SAFEs.

Recommendation

Disallow creating SAFEs for other users or require permissions when doing so. Alternatively, we recommend thoroughly documenting the current behavior of the `HaiSafeManager` in order to inform users that the list of SAFEs from a user may not represent only SAFEs created or owned by them.

19. Mismatch between documentation and implementation

On the documentation website, some functions are described as Authorized, while in the implementation codebase, they are not restricted. These functions are `popDebtFromQueue` and `cancelAuctionedDebtWithSurplus` from `AccountingEngine`, and `restartAuction` and `terminateAuctionPrematurely` from `DebtAuctionHouse`. After further inspection, they do not seem to pose any issues by not requiring access controls, which is reiterated by the fact that on GEB these functions are also unrestricted.

Recommendation

We recommend updating the project documentation to match the implementation.

20. Typos

In `src/contracts/utils/Authorizable.sol:11`, the NatSpec reads “Authorization control is boolean and handled by `onlyAuthorized` modifier”. In reality, there is no `onlyAuthorized` modifier, but rather an `isAuthorized` modifier.

In `ICollateralAuctionHouse` line `234`, the parameter name `_initialBidder` is used instead of the correct one `_auctionIncomeRecipient` that is used on the implementation.

Recommendation

We recommend updating the documentation to match the contract implementation.

21. Unused return values

In `src/contracts/proxies/GlobalSettlementActions.sol:37`, the `_safe` local variable is updated by a `_safeEngine.safes` external call but this return value is not used by the function.

Recommendation

We recommend removing this unnecessary external call.

22. Incorrect parameter emitted on events

In `src/contracts/CollateralAuctionHouse.sol:360`, the `terminateAuctionPrematurely` function emits a `TerminateAuctionPrematurely` event to log certain parameters when an auction is prematurely terminated. The third parameter is `_leftoverReceiver`, and is passed as the `_auction.forgoneCollateralReceiver`. The issue is that this function calls `safeEngine.transferCollateral` with destination equals to `msg.sender`, which is not necessarily the forgone collateral receiver, depending on which account is calling this function.

In `src/contracts/LiquidationEngine.sol:219`, the `liquidateSAFE` function emits a `Liquidate` event to log certain parameters when a SAFE is liquidated. The fifth parameter is named `_amountToRaise` according to `ILiquidationEngine`, which is "the Amount of system coins to raise in the collateral auction [rad]". The issue is that in the function implementation, the emitted parameter is simply `_limitAdjustedDebt * _safeEngCData.accumulatedRate`, which does not include the `__cParams.liquidationPenalty` that is multiplied by this result to generate the `_amountToRaise` local variable. As a result, it does not equal the total amount of system coins required in the collateral auction, that is passed to the `startAuction` function from `ICollateralAuctionHouse`. This same mistake appears to exist on the GEB codebase.

Recommendation

We recommend updating the parameters of these events so that off-chain monitoring systems properly the leftover receiver in all cases.

23. Missing revert string on `HaiProxyRegistry` function

In `src/contracts/proxies/HaiProxyRegistry:50`, the _build function contains a check to prevent the creation of a new `_proxy` if the user already has one and remains the owner. Nevertheless, this `require` does not contain a revert string, which can be confusing to the caller.

Recommendation

We recommend adding a revert string to this `require` check. Alternatively, we recommend using Solidity custom errors in the same way as applied in the rest of the codebase. In this case, consider refactoring `CollateralBidActions` to replace the `require` checks in order to maintain the same coding style throughout the protocol.

24. Protocol does not support rebasing/fee-on-transfer tokens

In `src/contracts/proxies/actions/CommonActions.sol` and `src/contracts/utils/CollateralJoin.sol`, the contracts perform safeTransfer and safeTransferFrom calls on external ERC20 smart contracts for the collaterals accepted by the HAI system. The issue is that some rebasing or fee-on-transfer tokens may not hold the expectation that the transferred amount from the sender address will match the received amount by the destination address. As a result, the internal accounting system may break.

Recommendation

We recommend refactoring these contracts to adapt to rebasing or fee-on-transfer tokens if these are intended to be supported by the project. Alternatively, properly document that these types of tokens are not supported by HAI and thoroughly vet each collateral token before integrating them into the protocol.

25. Possible reentrancy if ERC777 tokens are used as collateral

In `CollateralJoin` contract, the function `exit`, the CEI pattern is not used, if at any time the system would be to incorporate any ERC777 tokens as collateral, there could be a reentrancy risk due to the transfer function being called before modifying the collateral balance of the account first.

Recommendation

To address this risk, we recommend following the CEI pattern in the function or using a nonReentrant modifier in the case a ERC777 is used as collateral for the protocol.

26. Missing `whenEnabled` modifier

In `DebtAuctionHouse` line `123`, the function `restartAuction` does not implement the modifier `whenEnabled` which can lead to an auction restarting after settlement.

Recommendation
We recommend adding the mentioned modifier to the function in order to prevent any rare edge cases.

27. Unsafe cast to int256

In `GlobalSettlement` line `281`, the variable `_minCollateral` is converted to `int256` without checking possible overflows. This could lead to wrong values being used.

Recommendation
We recommend using the method `.toInt()` as on any of the other values converted.

Appendix

As part of our security review, we have implemented a fuzz testing campaign to validate the mathematical properties of the function `Math.rpow`, which is particularly complex due to the usage of inline assembly. Since we were not able to find any major issues with the function implementation, we include the tests in the appendix as a reference for the HAI protocol team to improve its test coverage. We recommend including it in the project repository to ensure that no regressions are introduced if the library is ever changed.

Note: a loss of precision of the order of 1 basis point was found for some associative exponential operations when the base is of the order of `RAY/10`. We believe this is expected because of the rounding errors incurred by the exponential operation. We could not find any counterexamples after increasing the minimum bound parameter in function base to `RAY*9/10`.

Test case 1

```
// SPDX-License-Identifier: GPL-3.0
// test/unit/Math.t.sol
// forge test --match-path test/unit/Math.t.sol -vv --fuzz-runs 100000
pragma solidity 0.8.19;

import "forge-std/Test.sol";
import "../../src/libraries/Math.sol";

//
https://github.com/crytic/properties/blob/main/PROPERTIES.md#abdkmath64x64
contract MathTest is Test {
    uint256 private constant PRECISION = 1;
    uint256 private constant PERCENT = 100_00;

    function setUp() public {}

    function test_pow_test_zero_exponent(uint256 x) public {
        assertEq(Math.rpow(x, 0), RAY, "Power of zero should be one.");
    }

    function test_pow_test_zero_base(uint256 x) public {
        assertEq(
            Math.rpow(0, base(x)),
            0,
            "Zero to the power of any number should be zero."
        );
    }

    function test_pow_test_one_exponent(uint256 x) public {
        x = base(x);

        assertEq(
            Math.rpow(x, 1),
            x,
            "Power of one should be equal to the operand."
        );
    }

    function test_pow_test_base_one(uint256 a) public {
        assertEq(
            Math.rpow(RAY, a),
            RAY,
            "Any number to the power of one should be equal to the
operand."
        );
    }

    function test_pow_test_product_same_base(
        uint256 x,
        uint256 a,
        uint256 b
    ) public {
        x = base(x);
        a = exponent(a);
        b = exponent(b);

        uint256 x_a = Math.rpow(x, a);
        uint256 x_b = Math.rpow(x, b);
        uint256 x_ab = Math.rpow(x, a + b);

        assertApproxEq(
            Math.rmul(x_a, x_b),
            x_ab,
            PRECISION,
            "Product of powers of the same base property."
        );
    }

    function test_pow_test_power_of_an_exponentiation(
        uint256 x,
        uint256 a,
        uint256 b
    ) public {
        x = base(x);
        a = bound(a, 0, 8);
        b = bound(a, 0, 8);

        uint256 x_a = Math.rpow(x, a);
        uint256 x_a_b = Math.rpow(x_a, b);
        uint256 x_ab = Math.rpow(x, a * b);

        assertApproxEq(
            x_a_b,
            x_ab,
            PRECISION,
            "Power of an exponentiation property"
        );
    }

    function test_pow_test_distributive(
        uint256 x,
        uint256 y,
        uint256 a
    ) public {
        x = base(x);
        y = base(y);
        a = exponent(a);

        uint256 x_y = Math.rmul(x, y);
        uint256 xy_a = Math.rpow(x_y, a);
        uint256 x_a = Math.rpow(x, a);
        uint256 y_a = Math.rpow(y, a);

        assertApproxEq(
            Math.rmul(x_a, y_a),
            xy_a,
            PRECISION,
            "Distributive property for power of a product"
        );
    }

    function test_pow_test_values(uint256 x, uint256 a) public {
        x = base(x);
        a = bound(a, 1, 16);

        uint256 x_a = Math.rpow(x, a);

        if (x >= RAY) {
            assertGe(
                x_a,
                RAY,
                "Power result should increase or decrease (in absolute
value) depending on exponent's absolute value."
            );
        } else {
            assertLt(
                x_a,
                RAY,
                "Power result should increase or decrease (in absolute
value) depending on exponent's absolute value."
            );
        }
    }

    // Test for abs(base) < 1 and high exponent
    function pow_test_high_exponent(uint256 x, uint256 a) public {
        x = bound(x, 0, RAY - 1);
        a = bound(a, 0, 2 ** 64);

        uint256 result = Math.rpow(x, a);
        assertEq(
            result,
            0,
            "Power edge case: Result should be zero if base is small and
exponent is large."
        );
    }

    function assertApproxEq(
        uint256 a,
        uint256 b,
        uint256 delta,
        string memory message
    ) internal {
        uint256 max = (a > b) ? a : b;
        uint256 min = (a > b) ? b : a;
        uint256 diffPercentBase = max != min ? (max - min) * PERCENT /
((max + min) / 2) : 0;
        bool isApproxEq = diffPercentBase <= delta;
        console.log(a, b, diffPercentBase);
        assertTrue(isApproxEq, message);
    }

    function base(uint256 x) internal view returns (uint256) {
        // NOTE: changingthe minimum value to RAY/10 breaks
`test_pow_test_distributive` and `test_pow_test_product_same_base`
        return bound(x, RAY*9/10, 2*RAY);
    }

    function exponent(uint256 a) internal view returns (uint256) {
        return bound(a, 0, 16);
    }
}
```

Disclaimer