While answering to the audit we made some modifications to the already audited code and hereby request a second audit to the made changes.

Explanation of said changes and their motive:

1 - Implementation of protocol fees to mint:

a. When msg.sender desires to mint an amount, a fee is taken from his input amount, and only minted the amount - fee, sending said fee to a treasury address;

2 - Abstract StableEngine logic, to create a new engine contract (**RadiantEngine**) to keep basic logic correct with SenecaEngine.

a. Radiant Engine implements Radiant Wrapper, a 4626 vault standard that mints a 1155 token used as wrapper for rebasing tokens & tokens that increment balanceOf, like Radiant Capital **rTokens** (https://arbiscan.io/address/0x727354712BDFcd8596a3852Fd2065b3C34F4F770)
https://docs.radiant.capital/radiant/contracts-and-security/arbitrum-contracts

b. When minting in this engine, before accepting the rToken, it wraps it in 1155 token , both deposited in the engine contract, and makes management of accounting using erc20 and erc1155 logics.

3 - Burning fees, with compounded interest rate fees,

Repo: https://github.com/SenecaDefi/seneca-stablecoin

a. Changed mint() function to include timestamps and update current user debt with interest rate:
https://github.com/SenecaDefi/seneca-stablecoin/blob/23c7d082b9e7e8f354ea26b0592d35f02cf2f7b5/src/StableEngine.sol#L244

b. Changed burn() function to ask more stable to burn than current stored debt to incur for interest rate fee.
https://github.com/SenecaDefi/seneca-stablecoin/blob/23c7d082b9e7e8f354ea26b0592d35f02cf2f7b5/src/StableEngine.sol#L306

c. Cumulative interest rate calculated with Aave library , library does not need audit, only its usage within the protocol.
https://github.com/SenecaDefi/seneca-stablecoin/blob/23c7d082b9e7e8f354ea26b0592d35f02cf2f7b5/src/StableEngine.sol#L447

d. Expected mint and burn fees to be paid here and acquired from a secondary market / engine, if the user does not have enough funds with him.

Audit: https://solidity.finance/audits/SenecaStablecoin/

Finding #1

StableEngine - Low

Description: *Users are allowed to borrow as long as their resulting health factor would be above the liquidation threshold.*

Risk/Impact: *An inexperienced user could borrow at very marginally above the liquidation threshold. A subsequent price drop in their collateral would result in liquidation.*

Recommendation: *The team should consider adding a buffer to the health factor users are allowed to borrow at to prevent immediate liquidation.*

Answer:

- *We don't consider this a problem, as it only affects inexperienced users, who won't use the frontend dapp.*

Finding #2 - StableEngine - Informational

Description: *The getAccountCollateralValue() function loops through each supported collateral token and always calls the _getUsdValue() function even if the user has not deposited any amount of the respective token.*

```
function getAccountCollateralValue(address user) public view returns (uint256
totalCollateralValueInUsd) {
  for (uint256 index = 0; index < s_collateralTokens.length; index++) {
      address token = s_collateralTokens[index];
      uint256 amount = s_collateralDeposited[user][token];
      totalCollateralValueInUsd += _getUsdValue(token, amount);

  }
```

Recommendation: *The team could modify the getAccountCollateralValue() function to only call the _getUsdValue() function when `amount` is not equal to zero for additional gas savings on each call.*

Answer: *We implemented the recommendation at: https://github.com/SenecaDefi/seneca-stablecoin/blob/b41df97a84f52b3a470b86220f6aa402929dece3/src/StableEngine.sol#L409*

Finding #3 - StableEngine - Informational

Description: *The liquidate() function verifies the caller's health factor exceeds the minimum threshold at the end of the function, even though the function's logic doesn't alter the caller's health factor.*
Recommendation: *The `revertIfHealthFactorIsBroken(msg.sender);` logic could be repositioned to the beginning of the function for additional gas savings on each call.*

Answer: *The `revertIfHealthFactorIsBroken(msg.sender)` needs to be at the end of the code to ensure that after all the user's accounting changes it stays with a positive health factor.*

Finding #4 - Oracle & StableEngine - Informational

Description: *The `sequencerUptimeFeed` and `FEE_PERCENTAGE` state variables can only be set one time in the constructor but are not declared immutable.*

Recommendation: *The above state variables could be declared immutable for additional gas savings on each reference.*

Answer: *changes were addressed, as we changed constructors to initalized functions;*

https://github.com/SenecaDefi/seneca-stablecoin/blob/master/src/engines/SenecaEngine.sol

Finding #5 - Oracle - Informational

Description: *The `_PERIOD_TIME` state variable can never be modified but is not declared constant.*

Recommendation: *The above state variable could be declared constant for additional gas savings on each reference.*

Answer: *changes were addressed, as we changed constructors to initialize(...) functions;*

https://github.com/SenecaDefi/seneca-stablecoin/blob/master/src/engines/SenecaEngine.sol

---

Finding #6 - StableEngine - Informational

Description: *The `PRECISION`, `ADDITIONAL_FEED_PRECISION`, and `FEED_PRECISION` state variables are not used in the contract.*

Recommendation: *The team should either remove the above state variables to reduce contract size and deployment costs or utilize them in a way that fits their intended functionality.*

Answer: *changes were addressed as recommended;*