CERTORA

Formal Verification of Synthetix Multi-Collateral Loans

Summary

This document describes the specification and verification of Synthetix MultiCollateral loan using Certora Prover. The work was undertaken from December 10 - December 17, 2020. The latest commit that was reviewed and run through the Certora Prover was **3778aaf8f2d7cf5fd8c651a0cfd72001ea41d113**.

The scope was new contracts and code changes to support multi collateral loans. The relevant contracts are: Collateral, CollateralErc20, CollateralEth, CollateralManager, CollateraManagerState, CollateralState and MultiCollateralSynths.

The Certora Prover proved the implementation of Synthetix correct with respect to the formal rules written by the Certora teams. During the verification process, the Certora Prover and the team's manual review discovered a number of bugs in the code listed in the table below. The next section formally defines high-level specifications of the Multi-Collateral contracts.

List of Main Issues Discovered

	Rule broken	Description	Severity	Mitigation
Loss of system fees	Additivity of open() and draw()	Fees are only paid on open() and not on draw() of a loan, so by opening and then drawing one can avoid paying fees	Medium	Fixed
Unauthorized changes	Privileged Operation	The following functions are not privileged: updateRates, updateBorrowRates, incrementLongs	High	Fixed independently by Synthetix
Loss of user funds	Total assets of a user is preserved	Miscalculation in the closure of a short loan causes the user to get only part of the collateral	Medium	Fixed

Lock of user funds		Unable to close short loans when holding multiple short loans per collateral-token/loaned-token pair	Medium	Fixed

Disclaimer

The Certora Prover takes as input a contract and a specification and formally proves that the contract satisfies the specification in all scenarios. Importantly, the guarantees of the Certora Prover are scoped to the provided specification, and any cases not covered by the specification are not checked by the Certora Prover.

We hope that this information is useful, but provide no warranty of any kind, express or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

Notations

1. ✔ indicates the rule is formally verified on the latest commit. We write ✔* when the rule was verified on a simplified version of the code (or under some assumptions), where exchange rates are assumed to be constant.
1. ✖ indicates the rule is violated in the version under test.
2. ✍ indicates the rule is not yet formally specified.
3. 🔁 indicates the rule is postponed due to time limitation

Verification of CollateralERC20

Properties

1. **Integrity of loan Owner ✔**
   `getLoan(owner, loanId).account == owner`

2. **Integrity of open ✔***
   open creates a loan of `x` synths, leading to synth balance increase by `x-fee`.

3. **Integrity of draw ✔***

   `draw(loanId,x)` increases the amount of loan `loanId` by `x`.

4. **Additivity of draw ✔***

   draw can be performed in two consecutive steps, or in one step:
   `draw(loanId, x) ; draw(loanId, y) ~ draw(loanId, x+y);`

5. **Additive open and draw ✖ - see issues**

   Opening an amount of `X` and then drawing an amount of `Y` is equivalent to opening an amount of `X+Y`.

6. **Integrity of Deposit ✔***

   `deposit(borrower, loanId, x)` increases the collateral of a loan `loanId` belonging to `borrower` by `x`.

7. **Additivity of deposit ✔***

   A deposit to a loan `loanId` belonging to `borrower` can be performed in two consecutive steps, or in a single step.
   `deposit(borrower, loanId, x) ; deposit(borrower, loanId, y) ~ deposit(borrower, loanId, x+y);`

8. **Privileged operations ✖ - see issues**

   Every operation that is expected to be privileged with limited access, can be executed by at most one address.

9. **Total assets of a user is preserved ✔***

   For a user *u* with a single loan `loanId`, the total balance in the borrowed asset + the balance in collateral asset + the collateral stored in the loan minus the loaned amount in the borrowed asset with fees is invariant of all functions.

   We define total assets of user *u* with respect to borrowed asset *t* and collateral asset *a* and `loanId`:
   `totalAssets(t, a, u, loanId) ≡ t.balanceOf(u) + a.balanceOf(u) + loanCollateral(u, loanId) - loanAmountWithFees(u, loanId)`

   ```
   { x = totalAssets(t, a, u, loanId)  }
         op;
   ```

```
{ totalAssets(t, a, u, loanId) = x}
```

Where `op` is an operation that affects the asset of at most one user