certora

Blend-Contract Formal Verification Report

Blend

January 2025

Prepared for
Script3 Ltd

certora

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform	Comment
blend-contracts	https://github.com/blend-capital/blend-contracts	895845f	Stellar	Formally verify user health check

Project Overview

This document describes the formal verification effort of Blend V1 using Certora's Sunbeam Prover which is a tool developed for verifying Soroban smart contracts. The tool operates at the Wasm bytecode level. This work was undertaken from **November 26, 2024**. We finished verifying the property on **January 15, 2025**.

In this document, we summarize how we formally verified that the user health factor is checked after actions are submitted for the Blend `pool`, as asked by the Blend team. The relevant code is in the `pool/src` directory of the Blend repository. The exact property we proved is discussed in detail below.

certora

Formal Verification

Formal Verification Property

All the verification-related code is in the `certora` branch of a fork of the Blend repo. It can be found here:

https://github.com/Certora/blend-contracts/tree/certora

Our specification language used for writing the properties can be found here:

https://github.com/Certora/solana-cvt/tree/dev-soroban

Repository Layout

We introduced the following:

- `pool/confs/pool.conf`:
  Configuration file for building and verifying the properties. To reproduce the results of this report, from `pool/confs` run `certoraRun pool.conf`

- `pool/src/spec`:
  Module containing specification-related definitions and properties. The main files are:
  - `rules.rs`: The property definitions.
  - `model.rs`: Auxiliary variables required for specifying the properties.
  - `summaries.rs`: Function summaries that we verify against (see below)
  - `token.rs`: A mock token implementation

We use Rust's feature mechanism (`#[cfg(feature = "certora")]`) to include the verification-specific code during compilation.

General Assumptions

- Loop iterations: Any loop was unrolled at most 2 times (iterations)

User Health

Summary. We specified a user-health property to check that a user's health is checked after every action is submitted. Ideally, we would prove that a given user's health factor is always greater than some nominal value. However, due to the complexity of the arithmetic involved, this direct property is inherently difficult for automatic provers. Instead, we specify a weaker property: if a user's positions change, then either (i) it is in a way that *obviously* preserves the health factor, or (ii) a designated function is called to *check* that the user is still "healthy."

Required changes. To prove this property, we needed to modify the Pool contract. First, as the property mostly depends on the `submit` entry point, we significantly refactored `actions::build_actions_from_request`. The refactoring adds a function for each request type. Next, we add summaries for each handler type and use these summary implementations in lieu of the summarized functions during the verification run.

Summarized code. The code for handling each request type is complex. To make the verification feasible, we introduce *summaries* for each request type in `pool/src/certora/summaries.rs`. We attempt to model the relevant behavior. It is important to note that not all of these summaries have been verified. Strictly speaking, we have not formally verified that they all soundly over-approximate the behavior of the summarized functions. However, all summaries are intended to be sound with respect to behaviors that affect the user health property (for example, we do not model writes to storage locations that are not needed for the property). Nevertheless, the correctness of the specification and verification results depends on the accuracy of these models.

The following is a high-level description of each summary:

- `build_supply, build_withdraw`
  Choose a reserve nondeterministically and set the user's supply of that reserve to an arbitrary value

- `Build_supply_collateral`
  Choose a reserve nondeterministically and *increase* the user's collateral of that reserve by an arbitrary positive value.

- `Build_withdraw_collateral`
  Choose a reserve nondeterministically and *decrease* the user's collateral of that reserve by an arbitrary positive value.

- `Build_borrow`
  Choose a reserve nondeterministically and *increase* the user's liability of that reserve by an arbitrary positive value.

- `Build_repay`
  Choose a reserve nondeterministically and *decrease* the user's liability of that reserve by an arbitrary positive value.

- `build_fill_user_liquidation_auction, build_fill_bad_debt_auction`
  Set the user's collateral and liabilities to new arbitrary values.

- `Delete_liquidation_auction`
  Do nothing.

- `Positions_hf_under`
  Set our model's flag saying that the designated "check" function has been called, and return an arbitrary boolean value.

- `Build_actions_from_request`
  Returns an arbitrary Actions, User state, and boolean value such that either the boolean is true (indicating that positions_hf_under should be called) or that the arbitrarily chosen User state's positions preserve the health of the original positions (e.g., simulating the effect of increasing collateral, decreasing liabilities, etc.).

`pool` crate: pool/spec/rules.rs:

Contract Properties

P-01 User health is checked for submitted actions	
Status: Verified	

Rule Name	Status	Description	Link to rule report
user_health	Verified	The main property: when execute_submit executes, then if a user's positions are updated, it is guaranteed that either a designated check health function has been called, OR the positions were updated in a way that the check is unnecessary (e.g., collateral strictly increased, liabilities strictly decreased).	report
user_health_sanity	Verified	Verifies that the user_health rule is not vacuously true.	report
build_actions_from_request	Verified	Verifies the soundness of a summary used in the verification of user_health	report
build_actions_from_request_sanity	Verified	Verifies that the build_actions_from_request rule is not vacuously true.	report

Appendix A: Recommendation for v2

While verifying the health check for v1 as described above, we observed that `build_actions_from_request` is designed to be a single monolithic function where each of the `Request` types is handled inside a `match`. To facilitate verification, we refactored it to have a dedicated function for each type of `Request`.

In general, modular code is better suited for formal verification and leads to better design.

We recommend that Blend's v2 code undergo a similar refactor so that future verification efforts will be smoother and the code will be easier to maintain and debug.

Disclaimer

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

Even though we hope this information is helpful, we provide no warranty of any kind, explicit 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.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.