StrataLabs

Clarity Contract Breakdown

Prepared For: **Granite**

Latest Review: **June 12th, 24'**

Latest Git Commit: **June 18th, 24'**

71b89d04eb588f26d3eb125af50905d47265f862

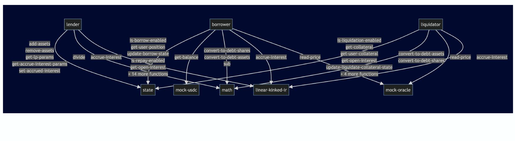
Abstract

The Granite lending protocol is a DeFi-lending market built on Stacks with the purpose of enabling overcollateralised loans using SIP-10 tokens.

The protocol is laid out over five (5) core contracts & five (5) helper contracts. There are three (3) major user-types (borrower, lender, liquidator) in addition to two (2) support user-types (governance & guardian).

Contract Table

Name	Type	Call?	Priv	Pub	Read
borrower	logic	33	8	4	1
bundler	helper	4	1	3	0
governance	governance	19	25	12	0
lender	logic	10	1	3	2
linear-kinked...	helper	0	9	1	6
liquidator	logic	18	1	3	2
math	helper	2	0	0	5
pyth-oracle	helper	2	4	2	0
state	logic	4	1	32	26



Findings

Below is a summary & list of all priority (0-3) issues found throughout an independent or paired review session. P0s have a high certainty in loss of protocol | user funds, P1s have a low probability through events such as admin accidents | governance exploits, P2s are highly recommend & p3 issues are optional (usually syntax/optimization based)

Finding	Severity	Status
Broken Taylor-Series Approximation	**p0**	acknowledged
Map Iteration Within A Fold Iteration	p3	acknowledged
Minor (get collaterals position) Repeat	p3	acknowledged
Same Data Fetched Multiple Times	p3	acknowledged
Inconsistent Error Numbering In .State	p3	acknowledged

Broken Taylor-Series Approximation — p0 — acknowledged

The (taylor-6 ...) function in the linear-kinked-ir contract is currently not working as expected. Due to overflow, this function attempts to approximate e^x through a six-item Taylor series. As such, we'd expect that passing in (1) would result in e^1 ~= 2.71828. This is currently not the case.

This function, which is a helper called throughout (accrue-interest ...), is indirectly called by many functions throughout the protocol, making this an urgent priority finding.

Map Iteration Within A Fold Iteration — p3 — acknowledged

In the borrower contract, the (borrow ...) function calculates the 'total-max-ltv' by first performing a 'map' operation, wrapped by a 'fold' operation. Both of these are expensive sequence iterative functions. Fold is quite flexible & can almost assuredly handle the logic found in 'iterate-collateral-value.

As discussed, the team hasn't optimize the contract yet & will take care of this when they do.

Minor (get collaterals position) Repeat — p3 — acknowledged

In the borrower contract, in both the (borrow ...) function, there is an issue with the use of (get collaterals position); it's uneccessarily set twice - in LOCs 38 & 41. In the latter function, we see (get collaterals position) called three times in LOC 125 - 127, when it should be stored as a variable once & then called afterwards.

As discussed, the team hasn't optimize the contract yet & will take care of this when they do.

Same Data Fetched Multiple Times — p3 — acknowledged

In the liquidator contract, the (liquidate-collateral ...) function fetches the same data from the .state contract three separate times. Get-collateral-value already fetches collateral so the following (collateral-info ...) is not necessary; additionally, (get-collateral-value) internally calls get-user-collateral which is also re-fetched & re-unwrapped in (user-balance ...) a few lines down. Same with (collateral-price ...).

As discussed, the team hasn't optimize the contract yet & will take care of this when they do.

Inconsistent Error Numbering In .State — p3 — acknowledged

The rest of the protocol throws four-digit uint error codes with a different beginning starting integer for each contract (in borrower.clar, 2xxx, in liquidator.clar, 3xxx). However, the state.clar contract throws strictly **three**-digit uint error codes. Minor inconsistency with no functional effect either way.