Granite Safety Module Audit Report

Version 1.0

Conducted by:
Alin Mihai Barbatei (ABA)

September 03, 2024

Table of Contents

1 Introduction

1.1 About ABA

ABA, or Alin Mihai Barbatei, is an established independent security researcher with deep expertise in blockchain security. With a background in traditional information security and competitive hacking, ABA has a proven track record of discovering hidden vulnerabilities. He has extensive experience in securing both EVM (Ethereum Virtual Machine) compatible blockchain projects and Bitcoin L2, Stacks projects.

Having conducted several solo and collaborative smart contract security reviews, ABA consistently strives to provide top-quality security auditing services. His dedication to the field is evident in the top-notch, high-quality, comprehensive smart contract auditing services he offers.

To learn more about his services, visit ABA's website abarbatei.xyz. You can also view his audit portfolio here. For audit bookings and security review inquiries, you can reach out to ABA on Telegram, Twitter (X) or WarpCast:

`https://t.me/abarbatei`

`https://x.com/abarbatei`

`https://warpcast.com/abarbatei.eth`

1.2 About Granite Safety Module

Granite is a DeFi lending market that offers overcollateralized loans on SIP-10 tokens managed and operated by immutable smart contracts. Granite was created and is managed by Trust Machines, a leading team of engineers, builders and researchers within the Stacks Bitcoin L2 ecosystem.

The newly added **Granite Safety Module** introduces LP tranches into the protocol, which allows Liquidity Providers to stake their LP tokens to earn a higher yield in exchange for being the first payers for bad debt.

Users stake their LP tokens and in return receive Staked-LP tokens, which are calculated and tracked using the standard share vault model.

Yields from repayments of debt are auto-compounded by the protocol by minting LP tokens into the Staked LP pool equal in value to the pool's share of a repayment. Staked LPs earn the base yield of unstaked LPs plus additional yield from staking.

This extension is inspired by *Aave Safety Module - Umbrella*. More on the *Granite Safety Module* component can be found in the internal documentation page `Safety module`.

1.3 Issues Risk Classification

The current report contains issues, or findings, that impact the protocol. Depending on the likelihood of the issue appearing and its impact (damage), an issue is in one of four risk categories or severities: *Critical*, *High*, *Medium*, *Low* or *Informational*.

The following table show an overview of how likelihood and impact determines the severity of an issue.

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behavior that's not so critical.

Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

Informational findings

Informational findings encompass recommendations to enhance code style, operations alignment with industry best practices, gas optimizations, adherence to documentation, standards, and overall contract design.

Informational findings typically have minimal impact on code functionality or security risk.

Evaluating all vulnerability types, including informational ones, is crucial for a comprehensive security audit to ensure robustness and reliability.

2 Executive Summary

2.1 Overview

Project Name	Granite Safety Module
Codebase	https://github.com/Trust-Machines/granite
Operating platform	Stacks
Programming language	Clarity
Audited commits	[1] 125263c31e5f98691baa2ee86cda30d3c0e20fbf [2] 3bf36227d35b1c76ee4f6d81c22de57afaa4a983 [3] e0a38838b9ff352920d7628251fd6891f6eb4a6c [4] 16280e97e80f28643c2a759caa71fd9552b02d3b
Remediation commit	9c3ffb40858a005e292de432582188c3d3f69289
Timeline	From 21.08.2024 to 28.08.2024 (8 days)
Audit methodology	Static analysis and manual review

2.2 Audit Scope

The audit focused on changes introduced with the *#188 Pull Request*. As such, the scope includes both changes to the existing contracts:

Modified files in scope

- `contracts/state.clar`
- `contracts/borrower.clar`
- `contracts/liquidator.clar`
- `contracts/liquidity-provider.clar`
- `contracts/governance.clar`
- `contracts/modules/math.clar`
- `contracts/modules/linear-kinked-ir.clar`

and also new contracts:

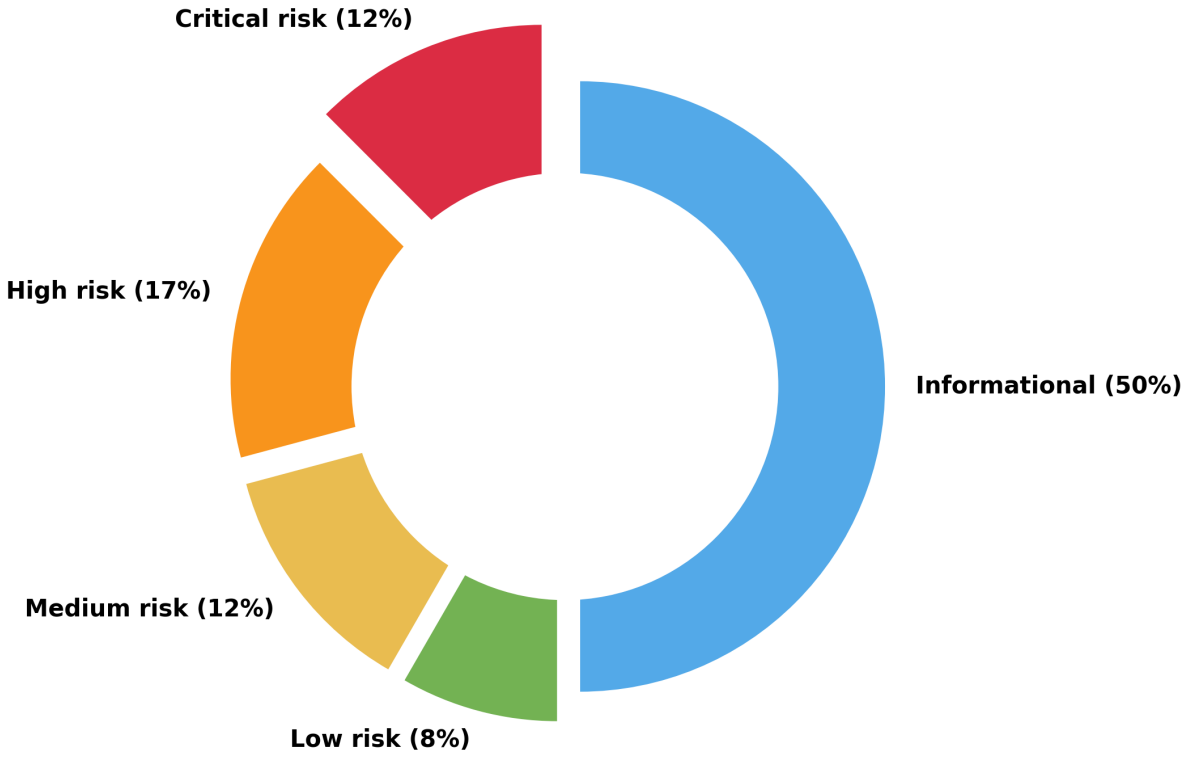
Files and folders in scope

- `contracts/staking.clar`

Files and folders in scope (continued)
- contracts/modules/staking-reward.clar

2.3 Summary of Findings

Severity	Total Found	Resolved	Partially Resolved	Acknowledged
Critical risk	3	3	0	0
High risk	4	4	0	0
Medium risk	3	3	0	0
Low risk	2	1	0	1
Informational	12	10	2	0



2.4 Findings & Resolutions

ID	Title	Severity	Status
C-01	Market open interest can be intentionally imbalanced	Critical	Resolved
C-02	Staking - State contracts can't be deployed due to cross referencing	Critical	Resolved
C-03	Stale price and oracle errors default to 0 price	Critical	Resolved
H-01	Granite Staked LP tokens generate reward while in unstaking period	High	Resolved
H-02	Interest accrual does not take into consideration staked open interest	High	Resolved
H-03	Borrowable balance incorrect updated without staked LP portion	High	Resolved
H-04	DOS on staking via inflation attack	High	Resolved
M-01	Missing governance multisig update-reward-params action	Medium	Resolved
M-02	Staking withdraw finalization period cannot be changed	Medium	Resolved
M-03	Debt cannot be socialized when there are no funds	Medium	Resolved
L-01	Granite Staked LP token URI cannot be changed	Low	Resolved
L-02	Users have no way of closing their initiated unstakes	Low	Acknowledged
I-01	Use consistent error code ranges	Informational	Resolved
I-02	Scaling factor can be removed from state contract functions return	Informational	Resolved
I-03	Reward rate documentation slight inconsistencies	Informational	Partially Resolved
I-04	state::socialize-user-bad-debt can be simplified	Informational	Resolved
I-05	Unnecessary extra whitespace characters	Informational	Partially Resolved
I-06	English dialect inconsistencies	Informational	Resolved
I-07	Internal token staking operations can be simplified	Informational	Resolved
I-08	Remove redundant padding comments	Informational	Resolved
I-09	staking::update-withdrawal-finalization-period lacks events	Informational	Resolved
I-10	state::get-add-collateral-params and state::get-remove-collateral-params can be simplified	Informational	Resolved
I-11	Return empty string or None for default token URI	Informational	Resolved
I-12	Inadequate event prints in sensitive operations	Informational	Resolved

3 Findings

3.1 Critical Severity Findings

[C-01] Market open interest can be intentionally imbalanced

> **Severity:** ***Critical risk*** *(Resolved)* *[PoC]*
>
> **Context:** [2] **:** ***state.clar****:699-701*

Description

When a users borrows from Granite, the amount he borrowed is added to the system as OI (open interest). Within the Granite market there are 3 components to the OI: the simple lenders (liquidity providers, LP), the staked LP providers (users that staked their initial LP tokens) and the protocol due OI.

Incorrectly, the borrowed amount is only added as OI to the LP holders and none to the LP stakers or protocol.

On borrow the LP OI is increased but when repaying, the repay amount is split into protocol, staked LP and normal LP.

This creates a sever issue where repaying and borrowing are not symmetrical in terms of accounting, and any borrow plus subsequent repay will imbalance the internal accounting.

Consider a scenario where a user borrows and immediately repays, (if done intentionally, in the same transaction). Because borrowing only increases the borrow OI and repaying decreases all 3, the simple action of borrowing plus repaying unbalances the OIs and moves any existing protocol OI to the LP providers OI while increasing the reserve balance.

In the mentioned scenario, an attacker can force all currently existing protocol OI to be directly paid to protocol reserve while moving the OI itself to the borrowers. The total OI remains the same but the internal assets are migrated to the protocol reserve.

Recommendation

After extensively discussing the issue with the team and following on their recommendation, it was decided that the best course of action, given current time and logistic constraints, is to split the OI repayment on accrued interest only. Meaning to not factor in the actual borrowed amounts (both per user and globally) when calculating the OI redistribution parts.

Thus repayments do not modify any OI if they are done in a shot span of time as no interest has accrued to be split. This resolves any imbalances between borrow and repay/liquidate/socialize operations.

Resolution: Resolved, the recommended fix was implemented in **PR#205**.

[C-02] Staking - State contracts can't be deployed due to cross referencing

> **Severity:** ***Critical risk*** *(Resolved)*
>
> **Context:** [⊶2] : ***staking.clar staking-reward.clar state.clar***

Description

The current overall staking logic that was added also includes a circular dependency that blocks the project from being fully deployed.

By circular dependency, the `state` contract has hardcoded the `.staking` contract and the `staking` contract has hardcoded the `.state` contract.

For deployment, each needs the other to exist, as such both will not deploy, leaving the protocol in a deadlock.

The issue was also identified by the team during testing.

Recommendation

To eliminate the cross reference move all the information required by the `state` contract in it, so that no reference to `staking` is done.

The `state` contract needed the `staking` contract for 2 role that can be mitigated:

A place to store the LP tokens that were staked by users

This is resolvable by transferring all LPs to the `state` contract itself, instead of storing it in the `staking` contract. Each stake operation would send LP tokens to the `state` contract, while each finalized unstaking would receive the LP tokens from the `state` contract itself.

Also, each function that requires knowing that amount, would simply query the `state` contract for it.

The `state` contract needed to know how much tokens were actually staked, as a number

The LP balance of the `state` contract cannot be directly used due to issues described in finding *Granite Staked LP tokens generate reward while in unstaking period* and *DOS on staking via inflation attack*.

Because a more complicated `get-actually-staked-LP-tokens` type of function must exist, that accounts for the mentioned issues, a relatively simple approach is to have that function within the `staking` contract, along side any other state elements required specifically for that point.

Whenever any actions done by users that modify the total staked LPs are done, besides noting it in the `staking` contract, also mirror (updated) it in the `state` contract.

By doing it like so, the `state` contract does not need to query the `.staking` contract any more to determine the total staked tokens.

The exact implementation details for the solution may differ. The following PR is a rough draft/example of how one such implementation may look like.

Resolution: Resolved. A fix was implemented in **PR#204**.

ABA: The team opted to implement a solution where borrower and liquidator pass the staking contract along to the state contract. While it does solve the issue at hand it increases code coupling and the difficulty of doing a contract migration. Team does not intend to have the staking contract as upgradable, thus it poses no issue to the project.

[C-03] Stale price and oracle errors default to 0 price

> **Severity:** ***Critical risk*** *(Resolved)*
>
> **Context:** [-o-4] : ***borrower.clar***:224 ***liquidator.clar***:225 ***math.clar***:74

Description

When the full Granite system will be up, users will use pyth to update the prices. It will be optional to call all user entry points with a pyth payload, as such, users may reach a case where a stale price is used.

Stale price is defined of the team (to be added post Nakamoto) and checked on each price retrieval with a `ERR-PYTH-PRICE-STALE` revert if it is stale:

```
(asserts! (not (is-stale? timestamp)) ERR-PYTH-PRICE-STALE)
```

However, within Granite, almost all price retrievals default to 0 value when an error occurs with the oracles, instead of halting execution:

- when having the collateral priced at any borrow
- when having the collateral priced at any liquidation
- in both liquidations and borrowing when retrieving the market token value via `math::get-market-asset-value`

In all these case, the default behavior would be lead to a 0 valuation and issues with having collateral possibly lost due to under valuation (at liquidations) or not being able to borrow.

Recommendation

To fix the issue execution must revert when the price is stale and any other oracle issues.

The simples fix is to change the `unwrap!` with an `unwrap-panic` in all the mentioned locations. This hides the error message though.

A more complex, but better version would be to modify the code to gracefully revert with an error message, similar to how it is done in liquidate-collateral.

Resolution: Resolved, the recommended fix was implemented in **PR#210**.

3.2 High Severity Findings

[H-01] Granite Staked LP tokens generate reward while in unstaking period

> **Severity:** *High risk* *(Resolved)*
>
> **Context:** [2] : *staking-reward.clar:74* *state.clar:855* *staking.clar:90*

Description

Liquidity providers can deposit their LPs to the safety module staking contract to earn more reward while taking up the risk of losing assets because of slashing due to bad debt payments.

To prevent an opportunistic staker from unstaking his LP tokens before a major bad debt liquidation, a wait period was introduced between the time when a user initiates his withdraw and when he can actually withdraw.

However, while the withdraw is pending, users still receiver rewards associated to it.

The issue lies with the way the `staking-reward` contract determines the amount of staked LP tokens. It does so by simply checking the balance of the `staking` contract.

```
(staked-lp-tokens (unwrap! (contract-call? .state get-balance .staking)
    ERR-LP-TOKEN-USER-BALANCE))
```

Without taking into consideration the amount of LP tokens to be unstaked, an abuser can:

- stake LP tokens
- instantly initiate an unstaking (maybe divide the whole amount in separate, smaller amounts)
- after the unstaking wait period ends, the user does nothing
- during this time, he will earn rewards and at the same time, can immediately exit if he sees any large liquidation that creates bad debt, thus bypassing the withdraw mechanism itself

Recommendation

In the `staking` contract, change the unstaking logic such:

- when initiating an unstake, directly convert the staked LPs to normal LPs and note them in the (repurposed) `user-total-unfinalized-withdrawal-amount`
- this amount will be released to the user when staking is finalized
- during the wait, the pending LPs must be disregarded from the number of LPs that was staked, as to not accrue any reward via normal vault share appreciation
- by doing it like so, you can simply the `transfer` function because the stakedLPs will be burned on unstaking initialization, users already loses them, so no need to further account them.
- also, track the total unfinalized withdrawal amounts in LP tokens, not stakedLP tokens, and use that when determining the amount of LPs that were staked.

Thus `get-balance` cannot be used to determine the number of staked LPs, needing a separate function, will refer to it as `total-staking-amount`.

This `total-staking-amount` must also be then called everywhere the number of LPs that were staked is needed, specifically in the `staking` contract, the `state::socialize-user-bad-debt` function and `staking-reward::calculate-staking-reward-percentage` function.

Resolution: Resolved, the recommended fix was implemented in **PR#207**.

[H-02] Interest accrual does not take into consideration staked open interest

> **Severity:** ***High risk*** *(Resolved)*
>
> **Context:** [-o-2] : ***linear-kinked-ir.clar***:*90*

Description

When accruing interest, the compounded interest is calculated based on total open interest in the market.

However, in the `linear-kinked-ir::accrue-interest` function, the open interest that is used to compute the interest increase incorrectly fails to take the staked LP open interest

```
(open-interest (+ lp-open-interest protocol-open-interest))
```

Because of this, the interest will always increase with less value than required.

Recommendation

Add the `staked-open-interest` variable to the `open-interest` before calculating the total interest in the `linear-kinked-ir::accrue-interest` function.

Resolution: Resolved, the recommended fix was implemented in **PR#200**.

[H-03] Borrowable balance incorrect updated without staked LP portion

> **Severity:** ***High risk*** *(Resolved)*
>
> **Context:** [-o-2] : ***state.clar***:*809,730*

Description

The available amount to be borrowed by users is capped both on the actual existing tokens in the `state` contract and on a `borrowable-balance` variable. This is done in order to prevent donation attacks.

The borrowable balance is, however, incorrectly updated without taking into consideration stacked LP amount. This results in users being unable to borrow amounts that are rightfully available for borrowing. The issue is present when liquidating a position or repaying debt.

Recommendation

In the `state::update-repay-state` and `state::update-liquidate-collateral-state` functions, increment the `borrowable-balance` variable with both the LP amount and the staked LP amount.

When borrowing, in the `state::update-borrow-state` function, the `borrowable-balance` is currently incremented with the entire borrowable amount. This is correct, however, if the modification done with regards to the *Market open interest can be intentionally imbalanced* issue will add a LP part, it must not be added to borrowable amount.

Resolution: Resolved. A fix was implemented in **PR#206**.

It will be fully completed as the C-01 and C-02 are solved.

[H-04] DOS on staking via inflation attack

> **Severity:** ***High risk*** *(Resolved) [PoC]*
>
> **Context:** [2] : ***staking.clar***:*73-84*

Description

The currently implemented staking logic allows for a slight variation of the first depositor attack with reduced damage to users in the form of a DOS (denial of service) on staking.

The attack would be done by a threat actor as:

- be the first to deposit and deposit a lot of market tokens in Granite, e.g. 100,000 USDC and get 100,000 LP shares
- stake 1 unit of LP tokens via the `staking::stake` function
- directly transfer the other entire amount, the remaining 99,999.99999999 USDC
- at this point, in a normal inflation attack, anyone else trying to stake would receive 0 tokens if their LP is less then 100,000 units.
- however, the f-mint? call fails when staking because of attempting to mint 0 tokens

```
(try! (ft-mint? staked-lp-token staked-lp-tokens-to-mint contract-caller))
```

Because of the `ft-ming?` revert in the `staking::stake` function, the malicious effect resulted by an attacker is to deny anyone the opportunity of depositing tokens until someone deposits the equivalent to the `.staking` contract LP, increasing the barrier of entry for new participants.

Users also lose due to rounding error if they don't stake multiples of the currently existing protocol LP balance. This brings up a further extension to the attack:

- second user wants to also stake and has 100,000 units, so that his `stake` action generates a share, to benefit from the extra yield
- attacker front-runs the user's stake with a direct transfer of 1 unit.
- now, since the user's stake is 100,000 but the total LP in the contract is 100,001, he is DOS'ed again.
- as such, the only way for other users to actually participate is to stake with an extra margin, losing it due to rounding, so that to be able to not be front-run by the first depositor.

The overall effect is a reduction in the efficiency of the staking system, allows for a unfair advantage to the first depositor, raises the barrier of entry to the staking contract and can cause user loss of funds.

Recommendation

Do not consider the current LP balance of the `staking` contract as the staked LP, but have a separate, internal accounting value for it.

Have this value incremented in three cases:

- when someone stakes via the `staking::stake`
- create a separate `increase-staked-balance` function, which is callable only by the `state` contract. Have this function be called after each LP mint to staking contract. Mints are done when `state::update-repay-state` is called and when `state::update-liquidate-collateral-state` is called
- have `staking::skim` value, callable by governance only, that will add the value with the difference between how much is present, the balance and how much is already directly accounted for both for staked and pending-unstaked amounts.

Also, have the internal balance value decrease in 2 cases:

- when someone unstakes via `staking::initiate-unstake`
- create a separate `decrease-staked-balance` function, which is callable only by the `state` contract. Have this function be called after each LP burn from the staking contract. This appears when debt is socialized in the `state::socialize-user-bad-debt`.

Resolution: Resolved, the recommended fix was implemented in **PR#207**.

3.3 Medium Severity Findings

[M-01] Missing governance multisig update-reward-params action

> **Severity:** *Medium risk* *(Resolved)*
>
> **Context:** [-o-1] : ***governance.clar***

Description

In the `staking-reward` contract, after initialization, only governance is allowed to change the staking rewards parameters via the `update-reward-params` function.

However, there is no support for such an action in the `governance` contract.

Without a supporting action, the settings for the staking reward module can never be changed after initialization.

Recommendation

Add an action that will allow `governance` to call the `staking-reward::update-reward-params` function.

Resolution: Resolved, the recommended fix was implemented in **PR#194**.

[M-02] Staking withdraw finalization period cannot be changed

> **Severity:** *Medium risk* *(Resolved)*
>
> **Context:** [-o-2] : ***staking.clar***:23

Description

In the newly-added `staking` contract, when a user initiates a withdrawal, he has to wait a designated time period (currently denoted in blocks) before being able to finalize the operation.

This time period is saved in the `withdrawal-finalization-period` data variable and set with an initial value of 100 blocks.

However, this variable cannot be changed as there are no setters or permissioned functions, as indicated by the project documentation:

> A withdrawals list will keep track of requests from LPs to withdraw and a governance-set cooldown period will be in place to prevent them from exiting just before a bad liquidation event gets settled.

Recommendation

Add permissioned logic needed to change the variable and create a governance action that facilitates this.

Resolution: Resolved, the recommended fix was implemented in **PR#201**.

[M-03] Debt cannot be socialized when there are no funds

> **Severity:** *Medium risk* *(Resolved)*
>
> **Context:** [2] : *state.clar:859*

Description

When socializing debt, at one point, funds are taken out of the total available assets in the market, from the `total-assets` variable.

In the extreme case that the entire total assets cannot cover the debt socializing, then the operation reverts with an underflow in the subtraction

```
(var-set total-assets (- (var-get total-assets) remaining-debt-to-socialize))
```

Realistically this situation would happen in an extreme price volatile situation and would lead to liquidations to fail if they are severely underwater and no assets to pay the debt. This will not allow the system to clean itself of debt to restart and would need manual debt repay before attempting liquidations.

Recommendation

Allow debt socialization to occur even if the remaining debt to be socialized is more then the total assets. Simply set the total assets to 0 in that case.

Resolution: Resolved, the recommended fix was implemented in **PR#209**.

3.4 Low Severity Findings

[L-01] Granite Staked LP token URI cannot be changed

> **Severity:** *Low risk* *(Resolved)*
>
> **Context:** [2] : *staking.clar*:20,69-71

Description

In the `staking` contract, the SIP compliant `get-token-uri` function returns the value of the `token-uri` variable. This value, however, is initially set to none and cannot be changed.

Recommendation

If the describe behavior is intended, modify `get-token-uri` to simply return none and remove the `token-uri` variable. Otherwise create a permissioned setter for the `token-uri` variable and have a corresponding governance action change it.

Resolution: Resolved. A fix was implemented in **PR#198**.

ABA: The team decided to return a generic, hardcoded string value, and not implement a governance setter.

[L-02] Users have no way of closing their initiated unstakes

> **Severity:** *Low risk* *(Acknowledged)*
>
> **Context:** [2] : *staking.clar*

Description

When users that deposited into the `staking` contract want to unstake, they first must call `initiate-unstake` and, after a period of time has passed, they must call `finalize-unstake`.

If, during this period, they change their mind and wish to continue their stake, there is no way for users to close their already initiated unstaking without waiting for the full finalization period to pass, finishing the unstaking and then subsequently staking again.

Recommendation

Create a function in the `staking` contract that allows users to cancel an initiated unstake.

Resolution: Acknowledged by the team.

Trust Machines: We prefer not to implement this functionality to prevent any possible gaming issues.

3.5 Informational Findings

[I-01] Use consistent error code ranges

> **Severity:** *Informational* *(Resolved)*
>
> **Context:** Global

Description

Through the codebase, there are error code ranges used in each contract that helps both clearly identify the error and from what contract did it came from.

Currently, the error ranges are not consistent. Some modules use thousands, while others use hundreds:

```
contracts\liquidity-provider.clar       1000-1999
contracts\borrower.clar                 2000-2999
contracts\liquidator.clar               3000-3999
contracts\governance.clar               4000-4999
contracts\meta-governance.clar          5000-5999
contracts\state.clar                     100-199
contracts\staking.clar                   700-799
contracts\constants.clar                None
contracts\modules\math.clar             None
contracts\modules\utility.clar          None
contracts\modules\linear-kinked-ir.clar  300-399
contracts\modules\pyth-oracle.clar       400-499
contracts\modules\staking-reward.clar    600-699
```

Another point to take into consideration is that some SIP error codes reach 100s values, as such, it is discouraged to use values within those ranges.

Recommendation

For all contracts, expect the `state` contract where the team intentionally wants to use the 100-199 range, use ranges in the thousands.

Resolution: Resolved, the recommended fix was implemented in **PR#203** and **PR#213**.

[I-02] Scaling factor can be removed from state contract functions return

> **Severity:** *Informational* *(Resolved)*
>
> **Context:** [-o-2] : ***state.clar***:*530,539*

Description

In the `state` contract, the `get-lp-params` and `get-debt-params` functions both return a `scaling-factor` which is never used.

Recommendation

Remove the `scaling-factor` return from the `get-lp-params` and `get-debt-params` functions.

Resolution: Resolved, the recommended fix was implemented in **PR#197**.

[I-03] Reward rate documentation slight inconsistencies

> **Severity:** *Informational* *(Partially Resolved)*
>
> **Context:** [-o-2] : *staking-reward.clar:122*

Description

In the reward rate documentation of the safety module, there are slight inconsistencies:

The function describing the reward rate (RR) shows that if the percentage of stacked LP is equal to the kink value, then the logic to calculate the slope is $slope_1 \cdot SP + RR_{\text{base}}$

However, in code, that branch is reached if it is only less, not also equal:

```
    (if (>= staked-percentage (var-get staked-kink))
      (staked-geq-kink staked-percentage)
      (staked-less-than-kink staked-percentage)
    )
```

Mathematically it makes no difference, as at that point both branches calculate the same value. There is only a slight inconvenience when cross checking the documentation with the implementation.

Another inconsistency is that the function graph that displays the reward rate has IR (interest rate) set as legend instead of RR (reward rate).

Recommendation

To proper align the documentation to the code, change the `>=` comparison in the `staking-reward.clar::calculate-reward-percentage` function to `>`. This would require to also change the name of the functions that calculate the RR as they contain `geq` and `less` in their name, to a more fitting value. An alternative is to change the documentation so that the equal (=) is on the second formula. This way no smart contract change is needed.

Also, change the `IR` legend line name to RR in the function graph.

Resolution: Partially resolved by the team.

Trust Machines: Will generate a better chart for the public documentation.

[I-04] state::socialize-user-bad-debt can be simplified

> **Severity:** *Informational* *(Resolved)*
>
> **Context:** [-o-2] : *state.clar:824*

Description

Within the `state::socialize-user-bad-debt` function, the open interest is retrieved as such, in a `let` declaration `(open-interest-data (get-open-interest))`.

Since the `state::get-open-interest` function call only retrieves the `open-interest` data variable, it can be directly retrieved without the need for an extra function call.

Recommendation

Retrieve the open interest via a `var-get open-interest` instruction instead of a `get-open-interest` call.

Resolution: Resolved, the recommended fix was implemented in **PR#203**.

[I-05] Unnecessary extra whitespace characters

> **Severity:** ***Informational*** *(Partially Resolved)*
>
> **Context:** Global

Description

Throughout the codebase there are instances of extra whitespace being present. These are unnecessary and increase code size.

Instances of extra whitespace cases:

- two spaces instead of one between code elements:
  - after the `calculate-reward-percentage` call in contracts/modules/staking-reward.clar#L83
  - before the `ERR-INSUFFICIENT-BALANCE` error in contracts/state.clar#L780
- unnecessary spaces between lines:
  - contracts/modules/staking-reward.clar#L4
  - contracts/modules/linear-kinked-ir.clar#L4
  - contracts/modules/linear-kinked-ir.clar#L138
  - contracts/governance.clar#L485
- extra space between ending set of brackets of parentheses and last word:
  - contracts/borrower.clar#L194
  - contracts/borrower.clar#L21
  - contracts/liquidity-provider.clar#L42
  - contracts/staking.clar#L26
- leading extra whitespace(s):
  - contracts/modules/linear-kinked-ir.clar#L193
  - contracts/modules/linear-kinked-ir.clar#L164
  - contracts/modules/linear-kinked-ir.clar#L154

- contracts/modules/linear-kinked-ir.clar#L150
- contracts/modules/linear-kinked-ir.clar#L146
- contracts/modules/linear-kinked-ir.clar#L75
- contracts/modules/linear-kinked-ir.clar#L23
- contracts/modules/pyth-oracle.clar#L14
- contracts/modules/pyth-oracle.clar#L45
- contracts/modules/pyth-oracle.clar#L54
- contracts/modules/pyth-oracle.clar#L60-L62
- contracts/modules/pyth-oracle.clar#L64-L65
- contracts/modules/pyth-oracle.clar#L80
- contracts/modules/staking-reward.clar#L21
- contracts/modules/staking-reward.clar#L30
- contracts/modules/staking-reward.clar#L107-L108
- contracts/modules/staking-reward.clar#L110-L111
- contracts/modules/staking-reward.clar#L115
- contracts/borrower.clar#L170
- contracts/liquidator.clar#L20
- contracts/liquidator.clar#L107
- contracts/state.clar#L78
- contracts/state.clar#L513
- contracts/state.clar#L521
- contracts/state.clar#L535
- contracts/state.clar#L577
- contracts/state.clar#L605
- contracts/state.clar#L860

Recommendation

Remove the mentioned extra whitespace characters.

Resolution: Partially resolved by the team in **PR#203**.

ABA: Only one of the mentioned cases of extra whitespace characters was fixed (the first mentioned case).

[I-06] English dialect inconsistencies

> **Severity:** ***Informational*** *(Resolved)*
>
> **Context:** [⊸2] **:** ***state.clar****:864*

Description

A common best practice is to use one language and dialect for the sake of consistency, readability and maintainability. Throughout the codebase American dialect is used with one exception of a British dialect version:

- the British *"socialising"* with *"s"* is used in a comment while the American *"socialize"* version with *"z"* is used in the actual function name (`socialize-user-bad-debt`).

Recommendation

Consider change *socialising* to *socializing* to maintain dialect consistency.

Resolution: Resolved, the recommended fix was implemented in **PR#203**.

[I-07] Internal token staking operations can be simplified

> **Severity:** ***Informational*** *(Resolved)*
>
> **Context:** [⟲2] : ***staking.clar***:*7-8,32,77,89,121*

Description

Within the `staking` contract, there are instances where the total staked LP balance or a user's balance is required. As the code is implemented now, these values are retrieved as inner function calls, instead of directly relying on the SIP `ft-get-balance` or `ft-get-supply` functions.

Recommendation

Modify the `staking` contract to optimize the code. Example:

```
-      (user-total-balance (unwrap! (get-balance sender)
   ERR-STAKED-LP-TOKEN-USER-BALANCE))
+      (user-total-balance (ft-get-balance staked-lp-token
   sender))
```

and

```
-      (total-staked-lp-tokens (unwrap! (get-total-supply)
   ERR-STAKED-LP-TOKEN-SUPPLY))
+      (total-staked-lp-tokens (ft-get-supply staked-lp-token))
```

Also, at this point the `ERR-STAKED-LP-TOKEN-USER-BALANCE` and `ERR-STAKED-LP-TOKEN-SUPPLY` errors can be removed from the file, and the rest of the errors (plus tests) modified accordingly

```
 ;; ERRORS
 (define-constant ERR-SENDER-MISMATCH (err u700))
 (define-constant ERR-LP-TOKEN-SUPPLY (err u701))
-(define-constant ERR-STAKED-LP-TOKEN-SUPPLY (err u702))
-(define-constant ERR-STAKED-LP-TOKEN-USER-BALANCE (err u703))
-(define-constant ERR-STAKED-LP-TOKEN-USER-NOT-ENOUGH-BALANCE
   (err u704))
-(define-constant ERR-MISSING-WITHDRAWAL (err u705))
-(define-constant ERR-WITHDRAWAL-NOT-FINALIZED (err u706))
-(define-constant ERR-ZERO-LP-TOKEN-STAKE (err u707))
```

```
-(define-constant ERR-ZERO-STAKED-LP-TOKEN-UNSTAKE (err u708))
+(define-constant ERR-STAKED-LP-TOKEN-USER-NOT-ENOUGH-BALANCE
   (err u702))
+(define-constant ERR-MISSING-WITHDRAWAL (err u703))
+(define-constant ERR-WITHDRAWAL-NOT-FINALIZED (err u704))
+(define-constant ERR-ZERO-LP-TOKEN-STAKE (err u705))
+(define-constant ERR-ZERO-STAKED-LP-TOKEN-UNSTAKE (err u706))
```

Resolution: Resolved, the recommended fix was implemented in **PR#203**.

[I-08] Remove redundant padding comments

> **Severity:** *Informational* *(Resolved)*
>
> **Context:** [-o-3] : ***governance.clar***:*744,763*

Description

In the `governance` contract, the `initiate-proposal-to-update-withdrawal-finalization-period` function is padded with redundant `;;;;;;;;;` comments.

Recommendation

Remove redundant padding comments.

Resolution: Resolved, the recommended fix was implemented in **PR#201**.

[I-09] staking::update-withdrawal-finalization-period lacks events

> **Severity:** *Informational* *(Resolved)*
>
> **Context:** [-o-3] : ***staking.clar***:*29-36*

Description

The `staking::update-withdrawal-finalization-period` function does not contain a `print` command, similar to other functions in this contract, that would allow for better off-chain monitoring.

Recommendation

Add a `print` call in `staking::update-withdrawal-finalization-period` that contains the action (function name), the old `withdrawal-finalization-period` and the new value.

Resolution: Resolved, the recommended fix was implemented in **PR#201**.

[I-10] state::get-add-collateral-params and state::get-remove-collateral-params can be simplified

> **Severity:** *Informational* *(Resolved)*
>
> **Context:** [2] : *state.clar*:633-657

Description

The `get-add-collateral-params` and `get-remove-collateral-params` functions from the `state` contract both declare 3 variables in a `let` declaration then return them. None of the variables are using the precedent declaration as such, they can directly be returned.

This completely removed the `let` declaration in both cases while still maintaining code readability.

Recommendation

Implement the mentioned simplification. Example implementation:

```
 (define-read-only (get-add-collateral-params (collateral-token
    principal) (user principal))
-  (let (
-      (collateral-info (unwrap! (map-get? collaterals
   collateral-token) ERR-COLLATERAL-NOT-SUPPORTED))
-      (user-balance (map-get? user-collaterals {user: user,
   collateral: collateral-token}))
-      (user-position (default-to {debt-shares: u0, collaterals:
   (list)} (map-get? positions user)))
-    )
-    (ok {
-      collateral-info: collateral-info,
-      user-balance: user-balance,
-      user-position: user-position,
-    })
+  (ok {
+    collateral-info: (unwrap! (map-get? collaterals
   collateral-token) ERR-COLLATERAL-NOT-SUPPORTED),
+    user-balance: (map-get? user-collaterals {user: user,
   collateral: collateral-token}),
+    user-position: (default-to {debt-shares: u0, collaterals:
   (list)} (map-get? positions user)),
+  }
 ))

 (define-read-only (get-remove-collateral-params
    (collateral-token principal) (user principal))
-  (let (
```

```
-      (collateral-info (unwrap! (map-get? collaterals
   collateral-token) ERR-COLLATERAL-NOT-SUPPORTED))
-      (user-balance (unwrap! (map-get? user-collaterals {user:
   user, collateral: collateral-token})
   ERR-INSUFFICIENT-BALANCE))
-      (user-position (unwrap! (map-get? positions user)
   ERR-NO-POSITION))
-    )
-    (ok {
-      collateral-info: collateral-info,
-      user-balance: user-balance,
-      user-position: user-position,
-    })
+  (ok {
+    collateral-info: (unwrap! (map-get? collaterals
   collateral-token) ERR-COLLATERAL-NOT-SUPPORTED),
+    user-balance: (unwrap! (map-get? user-collaterals {user:
   user, collateral: collateral-token})
   ERR-INSUFFICIENT-BALANCE),
+    user-position: (unwrap! (map-get? positions user)
   ERR-NO-POSITION),
+  }
 ))
```

Resolution: Resolved, the recommended fix was implemented in **PR#208**.

[I-11] Return empty string or None for default token URI

> **Severity:** *Informational* *(Resolved)*
>
> **Context:** [⊶4] : ***staking.clar***:80 ***state.clar***:464

Description

The SIP 10 standard defines that a token must implement a `get-token-uri` function. This function can return an optional string.

The Granite codebase returns a hardcoded `changeme` string. This is a redundant code size increase with no benefits.

```
(define-read-only (get-token-uri)
  (ok (some u"changeme"))
)
```

Recommendation

Return `(ok none)` to save space while also keeping compliance with the SIP, as the string is optional.

Resolution: Resolved, the recommended fix was implemented in **PR#210**.

[I-12] Inadequate event prints in sensitive operations

> **Severity:** *Informational* *(Resolved)*
>
> **Context:** [4] : ***liquidator.clar***:318-322 ***staking.clar***:137-166

Description

Within the codebase, there are instances where sensitive code is called and no extra events are printed (via the `print` command).

Instances:

- no prints are done in the `staking` contract in the functions `reconcile-lp-token-balance`, `increase-lp-staked-balance` and `decrease-lp-staked-balance`. Consider adding prints to these functions.
- very few data points are printed in the `liquidator::socialize-bad-debt` function. Consider adding more relevant information, such as `burned-staking-lp-tokens`, `lp-part staked-part`, `protocol-part` and `updated-total-borrowed-amount`.
-

Recommendation

Implement the mentioned changes.

Resolution: Resolved, the recommended fix was implemented in **PR#210**.

4 Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts the consultant to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. The consultant’s position is that each company and individual are responsible for their own due diligence and continuous security. The consultant’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology that is analyzed.

The assessment services provided by the consultant is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. Furthermore, because a single assessment can never be considered comprehensive, multiple independent assessments paired with a bug bounty program are always recommend.

For each finding, the consultant provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved, but they may not be tested or functional code. These recommendations are not exhaustive, and the clients are encouraged to consider them as a starting point for further discussion.

The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties. Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, the consultant does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

The consultant retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. The consultant is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. The consultant is furthermore allowed to claim bug bounties from third-parties while doing so.