ClarityAlliance

# GRANITE PYTH ORACLE UPGRADE SECURITY REVIEW

**Conducted by:**
KRISTIAN APOSTOLOV, ALIN BARBATEI (ABA)

DECEMBER 18TH, 2024

CONTENTS

# 1. About Clarity Alliance

**Clarity Alliance** is a team of expert whitehat hackers specialising in securing protocols on Stacks.

They have disclosed vulnerabilities that have saved millions in live TVL and conducted thorough reviews for some of the largest projects across the Stacks ecosystem.

Learn more about Clarity Alliance at clarityalliance.org.

CONTENTS

# 2. 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 Clarity Alliance 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 our 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. Clarity Alliance’s position is that each company and individual are responsible for their own due diligence and continuous security. Clarity Alliance’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 we agree
to analyze.

The assessment services provided by Clarity Alliance are 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. 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, Clarity Alliance does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

CONTENTS

# 3. Introduction

A time-boxed security review of the Granite’s Pyth Oracle Upgrade, where Clarity Alliance reviewed the scope and providing insight on improving the protocol.

# 4. About Granite

The Granite Protocol is an autonomous Bitcoin liquidity protocol where users can participate as liquidity providers, borrowers, or liquidators.

The protocol allows borrowers to take stablecoin loans using Bitcoin as collateral, without exposure to counterparty or rehypothecation risk. Liquidity providers can earn yield on stablecoins by providing liquidity to the pool, which is then lent to borrowers. Loans in Granite are best thought of as lines of credit, without set terms or repayment schedules. As long as the borrower maintains an adequate loan-to-value ratio (LTV), keeping their account in good health, they are not subject to liquidation. If a borrower’s LTV falls too low, a portion of their capital will be liquidated to bring their account back to solvency.

Granite enables BTC users to access DeFi without centralized custodians by leveraging Stacks’ Nakamoto upgrade and sBTC Bitcoin bridge.

# 5. Risk Classification

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

## 5.1 Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.

- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.

- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

## 5.2 Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.

- Medium - only a conditionally incentivized attack vector, but still relatively likely.

- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

## 5.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

## CONTENTS

# 6. Security Assessment Summary

### Review Commit Hash:

a881f0cdfb0852ac6817cb467c12723730470794

### https://github.com/Trust-Machines/granite/pull/236

- `contracts/borrower.clar`
- `contracts/governance.clar`
- `contracts/liquidator.clar`
- `contracts/modules/pyth-adapter.clar`

### https://github.com/Trust-Machines/granite/pull/239

- `contracts/modules/pyth-adapter.clar`

### https://github.com/Trust-Machines/granite/pull/249

- `contracts/modules/pyth-adapter-v1.clar`

### https://github.com/Trust-Machines/granite/pull/237

- `contracts/modules/linear-kinked-ir.clar`
- `contracts/state.clar`

### https://github.com/Trust-Machines/granite/pull/244

- `contracts/borrower-v1.clar`
- `contracts/liquidator-v1.clar`

### https://github.com/Trust-Machines/granite/pull/245

- `contracts/liquidator-v1.clar`

CONTENTS

# 7. Executive Summary

Over the course of the security review, Kristian Apostolov, Alin Barbatei (ABA) engaged with Trust Machines to review Granite. In this period of time a total of **10** issues were uncovered.

## Protocol Summary

| | |
|---|---|
| **Protocol Name** | Granite |
| **Repository** | https://github.com/Trust-Machines/granite |
| **Date** | December 18th, 2024 |
| **Protocol Type** | Lending Protocol |

## Findings Count

| Severity | Amount |
|---|---|
| Medium | 3 |
| Low | 1 |
| QA | 6 |
| **Total Findings** | **10** |

# Summary of Findings

| ID | Title | Severity | Status |
|---|---|---|---|
| [M-01] | Pyth Price Confidence Interval Is Not Validated | Medium | Resolved |
| [M-02] | Stale Price May Be Considered Valid | Medium | Resolved |
| [M-03] | Accruing Interest Does Not Account for Current Block | Medium | Resolved |
| [L-01] | Pyth Adapter Staleness Threshold Limit Is Not Validated | Low | Resolved |
| [QA-01] | Outdated or Ambiguous Pyth Adapter Documentation | QA | Resolved |
| [QA-02] | Simplification of Accrue Interest Operation | QA | Resolved |
| [QA-03] | Improve Contract Comments | QA | Resolved |
| [QA-04] | Implement a Bulk Read Prices Function | QA | Resolved |
| [QA-05] | Typographical Error | QA | Resolved |
| [QA-06] | Overlapping Error Codes With Dependencies | QA | Resolved |

CONTENTS

# 8. Findings

## 8.1. Medium Findings

## [M-01] Pyth Price Confidence Interval Is Not Validated

### Description

Prices provided by the Pyth Network include a degree of uncertainty, represented by a confidence interval.

Currently, the `pyth-adapter-v1` contract implementation only checks for price freshness and does not validate the confidence level. It is essential to validate the confidence level to ensure that the price returned by the network falls within an acceptable range for Granite.

For example, a price for `STX` might be `$3` with a confidence of `± $2`. In this scenario, the network is uncertain of the exact price, placing it within a `[$1, $5]` range.

Although such a case would be highly unusual, it is still possible and could lead to financial losses for users if this price is used in collateral evaluation.

### Recommendation

In the `pyth-adapter-v1` contract, implement a minimum confidence threshold (price/confidence), which can be updated by governance through a new action. This threshold should be checked when retrieving prices. Note that a confidence interval of 0 indicates no price spread and should be considered a valid price.

# [M-02] Stale Price May Be Considered Valid

## Description

The `pyth-adapter-v1` contract retrieves raw price data from the Pyth Network and subsequently validates its freshness.

```
(let ((block-timestamp (unwrap-panic (get-stacks-block-info? time
  (- stacks-block-height u1)))))
  (if (>= timestamp block-timestamp)
    true
    (> timestamp (- block-timestamp (var-get time-delta))))
)
```

It is evident that the timestamp for the last Stacks block is used, rather than the current one. This is because it is not possible to obtain the `stacks-block-height` information of the block in which a transaction is currently being processed. The timestamp is only generated after the block is committed to the chain.

This design flaw allows stale price updates to be used when updating prices in a block, which, if committed to the blockchain, would have a timestamp exceeding the stale limit.

In theory, Stacks blocks are minted every 5 seconds. However, real-time data shows variations of up to tens of seconds between blocks. Additionally, there are instances where the Stacks blockchain stops producing blocks and resumes after a significant delay. Prices would still be considered valid, as they are compared against the previous block's timestamp.

A real-life example is between Stacks blocks #242879 and #242880, which are 25 minutes apart. Any price validation done in block `#242880` would use block `#242879` 's timestamp, resulting in a 25-minute difference. Assuming a price staleness threshold of 15 minutes, a price updated 10 minutes before the end of block `#242879` would be considered valid in block `#242880` , despite being 35 minutes apart.

The impact is that prices updated between the staleness check limit and the current block commit would be considered valid, even if they are logically stale. This may lead to stale prices being used for collateral evaluation.

CONTENTS

## Recommendation

Currently, there is no mechanism to determine time-related information from code running in a transaction that is being executed in the latest block.

A workaround can be implemented using a variable to denote the Stacks block time and also consider it in the block timestamp when checking for staleness:

```
(define-constant STACKS_BLOCK_TIME u5)
(let ((block-timestamp (unwrap-panic (get-stacks-block-info? time
  (- stacks-block-height u1))))) STACKS_BLOCK_TIME))))
```

While the example snippet uses a constant 5 seconds to denote Stacks block time, a more robust implementation would involve making this value a variable that can be adjusted by governance. This would allow for adaptation to different scenarios and blockchain states.

CONTENTS

# [M-03] Accruing Interest Does Not Account for Current Block

## Description

In the `linear-kinked-ir-v1::accrue-interest` function, when calculating elapsed time, the current time is derived from the timestamp of the most recently added Stacks block:

```
(- (unwrap-panic (get-stacks-block-info? time (- stacks-block-height u1)))
```

This approach uses the timestamp of the last Stacks block rather than the current one. This is because it is not possible to obtain the `stacks-block-height` information for the block in which a transaction is currently being processed. The timestamp is only generated after the block is committed to the chain.

This design allows users to exit without paying interest for the time between the end of the last block and the execution of the current block.

For example, between Stacks blocks #242879 and #242880, which are 25 minutes apart, a user decides to fully repay their debt in block `#242880`. Since the accrual only considers the time up to the last block, 25 minutes of interest are lost, but only for this user as they exit the market.

At block `#242881`, the interest calculation would add the missing 25 minutes, only omitting the interest for the currently executing block (theoretically 5 seconds worth of interest).

Due to the way time is captured, users can time their market exit after a larger gap between blocks to reduce the interest paid.

This issue naturally arises in every operation that calculates interest, effectively acting as an interest lag and benefiting users.

## Recommendation

As mentioned in a similar issue, there is currently no mechanism to determine time-related information from code running in a transaction that is being executed in the latest block. A workaround can be implemented by using a variable to denote the Stacks block time and incorporating it into the block timestamp when determining elapsed time:

```
(define-constant STACKS_BLOCK_TIME u5)
(+ (- (unwrap-panic (get-stacks-block-info? time
  (- stacks-block-height u1))) STACKS_BLOCK_TIME)
```

Notes:

- While the example snippet uses a constant 5 seconds to denote Stacks block time, a more robust implementation would involve using a variable that can be adjusted by governance to account for different scenarios and blockchain states.
- In `linear-kinked-ir-v1::accrue-interest`, there are three instances where the current time is captured. If the function is not simplified, all instances will require this change.
- In `state-v1`, there are also two instances where the time is set, and the same logic must be applied in both cases.

# 8.2. Low Findings

## [L-01] Pyth Adapter Staleness Threshold Limit Is Not Validated

### Description

The `pyth-adapter-v1` allows governance to set a staleness threshold, `time-delta` . Currently, there is no validation to ensure that the provided time is neither too large nor too small.

For instance, setting it too low may result in continuous protocol reverts.

### Recommendation

When updating the `time-delta` value via `governance-v1::initiate-proposal-to-update-pyth-time-delta` , ensure that the time-delta value is validated to be above a minimum, reasonable value.

# 8.3. QA Findings

## [QA-01] Outdated or Ambiguous Pyth Adapter Documentation

### Description

The current Granite Pyth Adapter documentation contains sections that are either outdated or somewhat ambiguous.

1. Under the guidance of Trust Machines, the Stacks Pyth implementation has been updated, rendering the Staleness Check section outdated. There is now an API that checks for price freshness, and the Pyth developers have indicated that the 2-hour staleness check will be modified before launch.
2. The most current version of the Pyth implementation is no longer maintained by Hiro but by Trust Machines themselves.
3. The price feed map associates token principals with feed IDs, not tickers.
4. The `read-price` function has a parameter named `token`, not `ticker`.
5. The `update-price-feed-id` function has the first parameter as `(token principal)`, not `(ticker (string-ascii 10))`; additionally, the description incorrectly refers to it as a ticker.
6. The `update-time-delta` function does not specify the units for the time difference `delta`. It is in seconds, but this is not noted.
7. The description of the `is-valid` private function does not clearly define whether being **“within the governance-defined validity threshold”** includes accepting prices that are exactly at the delta limit. The implementation itself does not accept a price exactly at a delta limit value, only if it surpasses it.

### Recommendation

Address the ambiguities and update the outdated documentation entries mentioned.

CONTENTS

# [QA-02] Simplification of Accrue Interest Operation

## Description

The current implementation of `linear-kinked-ir-v1::accrue-interest` , which is invoked at all critical points, can be streamlined in several ways:

- The `if - else` logic, which includes a premature return in the true branch and a `let` declaration in the false branch, can be condensed into a single, direct `let` declaration with an `assets!` for the premature return.
- The elapsed block time is redundantly retrieved twice during a full `accrue-interest` call. By applying the first optimization, the `elapsed-block-time` variable can be initialized earlier and reused.
- The current time (`(unwrap-panic (get-stacks-block-info? time (- stacks-block-height u1)))`) is redundantly retrieved three times. Similar to the previous optimization, it should be retrieved only once and stored in a variable.

By implementing these optimizations, the smart contract execution costs will be reduced, and code readability will be enhanced.

## Recommendation

Implement the suggested changes.

Example of fee reduction if the proposed changes are applied:

```
(let (
    (time-now (unwrap-panic (get-stacks-block-info? time
      (- stacks-block-height u1))))
    (elapsed-block-time (- time-now last-accrued-block-time))
    (premature-return (asserts!
      (not (or (is-eq u0 elapsed-block-time) (not
        (contract-call? .state-v1 is-interest-accrual-enabled))))
      (ok {
        last-accrued-block-time: last-accrued-block-time,
        lp-open-interest: lp-open-interest,
        protocol-open-interest: protocol-open-interest,
        staked-open-interest: staked-open-interest,
        total-assets: total-assets
      })))
```

As observed through a `borrow` call:

| Cost | Before | After | Improvement | | --- | --- | --- | --- | | write_length | 498 | 498 | 0 | | write_count | 10 | 10 | 0 | | read_length | 455427 | 455324 | 103 | | read_count | 147 | 143 | 4 | | runtime | 848993 | 834678 | 14315 |

Improvements in runtime, read counts, and read length have been achieved.

# [QA-03] Improve Contract Comments

## Description

In the `governance-v1` contract, all maps used by the proposal system are documented, except for the newly added `update-time-delta-params` map. To maintain code uniformity, documentation should also be added to this map.

## Recommendation

Add the specified comment.

CONTENTS

# [QA-04] Implement a Bulk Read Prices Function

## Description

The current implementation includes a validation for critical operations that require collateral evaluation, ensuring that all prices must be available. This creates a pre-return scenario if there are issues with any collateral price.

The current approach is highly inefficient as it sequentially calls the `get price` function for each collateral.

```
(position (unwrap! (get user-position borrow-params) ERR-NO-POSITION))
;; ...
(position-collaterals (get collaterals position))
;; ...
(collateral-price-check (try!
  (fold check-collateral-price-exists position-collaterals (ok true))))

;; ...

(define-private (check-collateral-price-exists (collateral principal) (res
  (response bool uint)))
  (let (
    (result (try! res))
    (next-result (try! (contract-call? .pyth-adapter-v1 read-price collateral)))
  )
    (ok true)
  )
)
```

Each call to `pyth-adapter-v1::read-price` increases runtime costs in an already complex protocol.

## Recommendation

Implement a `bulk-read-price` function in the `pyth-adapter-v1` and utilize it to verify the existence of prices throughout the codebase.

# [QA-05] Typographical Error

## Description

The `check-collteral-price-exists` function, used throughout the codebase, contains a typographical error. The word `collteral` should be corrected to `collateral`.

## Recommendation

Correct the identified typo.

## CONTENTS

# [QA-06] Overlapping Error Codes With Dependencies

## Description

Granite contracts and Pyth contracts have overlapping error code ranges:

## Granite

```
state-v1:                 100
liquidity-provider-v1:    1000
borrower-v1:              2000
liquidator-v1:            3000
governance-v1:            4000
meta-governance-v1:       5000
staking-v1:               6000
linear-kinked-ir-v1:      7000
pyth-adapter-v1:          8000
staking-reward-v1:        9000
```

## Pyth

```
wormhole-core-v3:         1000
pyth-pnau-decoder-v2:     2000
pyth-oracle-v3:           3000
pyth-governance-v2:       4000
pyth-storage-v3:          5000
```

For example, invoking `borrower-v1::borrow` with a Pyth price feed payload that contains an invalid magic will trigger the error `pyth-pnau-decoder-v2:ERR_MAGIC_BYTES (err u2001)`. This error code is identical to `borrower-v1::ERR-INSUFFICIENT-FREE-LIQUIDITY (err u2001)`, which can cause confusion for any third-party integrator.

## Recommendation

Increase all Granite error code ranges to the tens of thousands, for instance, change `liquidity-provider-v1: 1000` to `10000`.