Security Assessment Report

Community Integrity Pool

September 11, 2024

Summary

The Sec3 team (formerly Soteria) was engaged to conduct a thorough security analysis of the Community Integrity Pool smart contracts.

The artifact was the source code of the following programs (excluding tests) and PRs in

- https://github.com/pyth-network/governance
- https://github.com/pyth-network/pyth-crosschain
- https://github.com/pyth-network/pyth-client
- https://github.com/pyth-network/pythnet

The initial audit focused on the following versions and revealed 8 issues or questions.

#	program	commit / PR
P1	integrity_pool	f5158d8ce4d61c2a06ab51fcf0e010ce18d8fedb
P2	staking	f5158d8ce4d61c2a06ab51fcf0e010ce18d8fedb
P3	publisher-caps	f5158d8ce4d61c2a06ab51fcf0e010ce18d8fedb
P4	pyth-crosschain PR#1778	pyth-crosschain PR#1778
P5	pyth-client PR#412	pyth-client PR#412
P6	pythnet PR#302	pythnet PR#302
P7	pyth-crosschain PR#1780	pyth-crosschain PR#1780

This report provides a detailed description of the findings and their respective resolutions.

Table of Contents

Result Overview

Issue	Impact	Status
INTEGRITY_POOL		
[P1-I-01] Inefficient "get_publisher_index" implementation	Info	Acknowledged
[P1-I-02] Function "advance" may use inaccurate "publisher_caps" data	Info	Acknowledged
[P1-I-03] Consider updating "delegation_record" before invoking "slash"	Info	Resolved
[P1-I-04] Missing removal mechanism for publishers	Info	Acknowledged
STAKING		
[P2-I-01] The "transfer_epoch" is never set	Info	Resolved
[P2-I-02] Position targeting "IntegrityPool" may still be credited into "target_account"	Info	Resolved
[P2-Q-01] The "UNLOCKED" positions in calculations	Question	Resolved
[P2-Q-02] Is the "unlocking_duration" always 1?	Question	Resolved

Findings in Detail

INTEGRITY_POOL

[P1-I-01] Inefficient "get_publisher_index" implementation

The "`get_publisher_index`" function of the "`integrity-pool`" program linearly iterates through the "`self.publishers`" array to locate a specified publisher, which is inefficient:

```
/* staking/programs/integrity-pool/src/state/pool.rs */
317 | pub fn get_publisher_index(&self, publisher: &Pubkey) -> Result<usize> {
318 |     for i in 0..MAX_PUBLISHERS {
319 |         if self.publishers[i] == *publisher {
320 |             return Ok(i);
321 |         }
322 |     }
323 |     err!(IntegrityPoolError::PublisherNotFound)
324 | }
```

This function is utilized in numerous operations within the "`integrity-pool`" program, and the frequent linear searches can lead to unnecessary CU cost overhead.

To address this inefficiency, we recommend keeping the "`self.publishers`" array sorted after every modification and modifying the "`get_publisher_index`" function to use a binary search to reduce time complexity.

Resolution

The team acknowledged this finding and clarified that not sorting "`self.publishers`" seems less error-prone, and currently, there are no compute unit issues.

INTEGRITY_POOL

[P1-I-02] Function "advance" may use inaccurate "publisher_caps" data

In the "advance" function, the "publisher_caps" data for the current epoch is used to create reward events for each epoch from "last_updated_epoch" to "current_epoch". However, using the current "publisher_caps" data to calculate events for past epochs is inaccurate. The "self_reward_ratio" and "other_reward_ratio" calculated from "publisher_caps" affect the final rewards that publishers and delegators receive, and this inaccuracy could potentially harm users' interests.

Although the "advance" instruction is permissionless, and the duration of a single epoch is relatively long (approximately one week), making it unlikely that no one will call "advance" during a epoch, it is still recommended to deploy an automated off-chain process to ensure that the "advance" instruction is called for each epoch.

```
/* staking/programs/integrity-pool/src/state/pool.rs */
171 | pub fn advance(
172 |     &mut self,
173 |     publisher_caps: &PublisherCaps,
174 |     y: frac64,
175 |     current_epoch: u64,
176 | ) -> Result<()> {

206 |     while i < MAX_PUBLISHERS && self.publishers[i] != Pubkey::default() {
207 |         let cap_index = Self::get_publisher_cap_index(&self.publishers[i], publisher_caps);
208 |
209 |         let publisher_cap = match cap_index {
210 |             Ok(cap_index) => {
211 |                 existing_publishers.set(cap_index);
212 |                 publisher_caps.get_cap(cap_index).cap
213 |             }
214 |             Err(_) => 0,
215 |         };

219 |         self.create_reward_events_for_publisher(
220 |             self.last_updated_epoch,
221 |             self.last_updated_epoch + 1,
222 |             i,
223 |             publisher_cap,
224 |         )?;

243 |         self.create_reward_events_for_publisher(
244 |             self.last_updated_epoch + 1,
245 |             current_epoch,
246 |             i,
```

```
247 |             publisher_cap,
248 |         )?;
```

Resolution

The team acknowledged this issue and clarified that they will run an automated service to call "advance" weekly.

INTEGRITY_POOL

[P1-I-03] Consider updating "delegation_record" before invoking "slash"

In the "integrity-pool" program, the "reward_program_authority" can create a slash event and invoke "slash" to reduce the amount of specified positions via a "staking" CPI.

```
/* staking/programs/staking/src/lib.rs */
799 | pub fn slash_account(
800 |     ctx: Context<SlashAccount>,
801 |     // a number between 0 and 1 with 6 decimals of precision
802 |     // TODO: use fract64 instead of u64
803 |     slash_ratio: u64,
804 | ) -> Result<(u64, u64)> {
...
856 | // position_data.amount >= to_slash since slash_ratio is between 0 and 1
857 | if position_data.amount - to_slash == 0 {
858 |     stake_account_positions.make_none(i, next_index)?;
859 |     continue;
860 | } else {
861 |     stake_account_positions.write_position(
862 |         i,
863 |         &Position {
864 |             amount:                position_data.amount - to_slash,
865 |             target_with_parameters: position_data.target_with_parameters,
866 |             activation_epoch:       position_data.activation_epoch,
867 |             unlocking_start:        position_data.unlocking_start,
868 |         },
869 |     )?;
870 | }
```

However, the "integrity_pool::slash" instruction does not verify if the "delegation_record" has been updated to the current epoch. Consequently, when "slash" is executed, the "delegation_record" may not accurately represent the current epoch. This becomes problematic when calculating rewards, as the "advance_delegation_record" function relies on "position.amount". If "advance_delegation_record" is called again within the same epoch, it will use the already slashed amount from the current epoch to update all rewards from the last advance to the current epoch, leading to inaccurately reduced reward calculations for previous epochs.

```
/* staking/programs/integrity-pool/src/state/pool.rs */
151 | event_amounts[last_event_index % MAX_EVENTS] += position.amount;
...
155 | for (i, amount) in event_amounts.iter().enumerate() {
156 |     let event = self.get_event(i);
157 |     let (delegator_reward_for_event, publisher_reward_for_event) = event.calculate_reward(
```

```
158 |         *amount,
159 |         publisher_index,
160 |         &self.publisher_stake_accounts[publisher_index] == stake_account_positions_key,
161 |     )?;
162 |
163 |     delegator_reward += delegator_reward_for_event;
164 |     publisher_reward += publisher_reward_for_event;
165 | }
```

It's recommended to ensure that the "`delegation_record`" is updated to the current epoch before invoking "`slash`".

Resolution

This issue has been resolved by PR#526.

INTEGRITY_POOL

[P1-I-04] Missing removal mechanism for publishers

In "`PoolData::advance`" function, if there is a new publisher in the "`publisher_caps`", it would be added to "`self.publishers`".

```
/* staking/programs/integrity-pool/src/state/pool.rs */
253 | for j in 0..(publisher_caps.num_publishers() as usize) {
254 |     // Silently ignore if there are more publishers than MAX_PUBLISHERS
255 |     if !existing_publishers.get(j) && i < MAX_PUBLISHERS {
256 |         self.publishers[i] = publisher_caps.get_cap(j).pubkey;
257 |         i += 1;
258 |     }
259 | }
```

However, in the current implementation, there is no instruction available to remove deprecated publishers from "`self.publishers`". The lack of a removal mechanism could result in a continuous accumulation of entries in "`self.publishers`", eventually causing it to exceed the "`MAX_CAPS`" limit.

Resolution

The team acknowledged this issue and clarified that the current number of publishers is far from 1024, so it won't be a problem for now.

STAKING

[P2-I-01] The "transfer_epoch" is never set

In the "update_voter_weight" instruction, if the "transfer_epoch" field stored in the metadata of the current stake account is later than the "epoch_of_snapshot" (which can either be the current epoch or the epoch in which the voting process for a proposal began, depending on the voting target), the update to the "voter_record" is rejected.

However, the "transfer_epoch" value has never been initialized.

```
/* staking/programs/staking/src/lib.rs */
472 | pub fn update_voter_weight(
473 |     ctx: Context<UpdateVoterWeight>,
474 |     action: VoterWeightAction,
475 | ) -> Result<()> {

567 |     if let Some(transfer_epoch) = ctx.accounts.stake_account_metadata.transfer_epoch {
568 |         if epoch_of_snapshot <= transfer_epoch {
569 |             return Err(error!(ErrorCode::VoteDuringTransferEpoch));
570 |         }
571 |     }
```

Resolution

This issue has been resolved by PR#524.

STAKING

[P2-I-02] Position targeting "IntegrityPool" may still be credited into "target_account"

In the "create_position" instruction, as long as "target_account" is not "None", the amount of the new position will be added to the total locked assets stored in "target_account". Even if the target type is not "Voting", such as in the case of an "IntegrityPool", this still allows the corresponding amount to be included in the total.

This can result in the recorded amount of locked assets for voting being higher than the actual amount, leading to lower voter weights for each individual in the subsequent calculations.

```
/* staking/programs/staking/src/lib.rs */
232 | if let Some(target_account) = maybe_target_account {
233 |     target_account.add_locking(amount, current_epoch)?;
234 | }
```

However, since positions targeting "IntegrityPool" require "pool_authority" as a signer, and the integrity-pool program currently does not include "target_account" when calling "create_position", this issue will not occur at this moment.

It is recommended to enforce that "target_account" must be "None" when the target is an "IntegrityPool".

Resolution

This issue has been resolved by PR#506.

STAKING

[P2-Q-01] The "UNLOCKED" positions in calculations

In "`utils::risk::validate`", the "`governance_exposure`" and "`integrity_pool_exposure`" of the positions are calculated by directly summing the amounts of positions with the same target.

```
/* staking/programs/staking/src/utils/risk.rs */
049 | for i in 0..stake_account_positions.get_position_capacity() {
050 |     if let Some(position) = stake_account_positions.read_position(i)? {
051 |         match position.target_with_parameters.get_target() {
052 |             Target::Voting => {
053 |                 governance_exposure = governance_exposure
054 |                     .checked_add(position.amount)
055 |                     .ok_or_else(|| error!(GenericOverflow))?;
056 |             }
057 |             Target::IntegrityPool { .. } => {
058 |                 integrity_pool_exposure = integrity_pool_exposure
059 |                     .checked_add(position.amount)
060 |                     .ok_or_else(|| error!(GenericOverflow))?;
061 |             }
062 |         }
063 |     }
064 | }
```

However, if a position is in the "`UNLOCKED`" state, its amount has been unlocked for the current epoch. We are considering whether such positions should be excluded from the calculations of "`governance_exposure`" and "`integrity_pool_exposure`".

Resolution

This issue has been resolved by PR#502.

STAKING

[P2-Q-02] Is the "unlocking_duration" always 1?

In "merge_target_positions", the "is_equivalent" function is used to determine whether two positions are equivalent. This function only checks whether the states of the two positions are the same in the current and previous epochs, without considering whether their future states will be the same.

This approach is appropriate when the state transitions always complete within a single epoch (e.g., "LOCKING"->"LOCKED", "PREUNLOCKING"->"UNLOCKING"), because once the current state is determined, the states for the following epochs can be inferred, and there is no need to check if the parameters set in the positions are the same.

However, for the state transition from "UNLOCKING" to "UNLOCKED", the duration of the transition depends on the "unlocking_duration" set in "GlobalConfig". If its value is greater than 2, two positions with the same current and previous epoch states might not be mergeable, as their UNLOCKED epochs could differ.

For example, if "unlocking_duration" is 3, and "position_1" was closed in epoch 1, entering the PREUNLOCKING state, while "position_2" was closed in epoch 2, their states would be as shown in the table below:

```
epoch:           1          2          3          4         5         6
position_1:  PREUNLOCK UNLOCKING UNLOCKING UNLOCKING UNLOCKED
position_2:            PREUNLOCK UNLOCKING UNLOCKING UNLOCKING UNLOCKED
```

As we can see, if we call "merge_target_positions" in epoch 4, "is_equivalent" will consider these two positions as equivalent, leading to a merge. However, their fully unlocked epochs are not the same, so they should not be merged.

We notice that the "integrity-pool" program directly uses the "UNLOCKING_DURATION" constant (with an actual value of 1) to call "get_current_position" to calculate the current state of the position, and the tests for staking program also set "unlocking_duration" to 1, so does the project assume that "unlocking_duration" must always be equal to 1 but not implementing the corre-

sponding checks to enforce this?

```
/* staking/programs/integrity-pool/src/utils/clock.rs */
010 | pub const UNLOCKING_DURATION: u8 = 1; // 1 epoch
/* staking/programs/integrity-pool/src/state/pool.rs */
094 | pub fn calculate_reward(

142 |             let position_state =
143 |                 position.get_current_position(event.epoch, UNLOCKING_DURATION)?;
/* staking/integration-tests/src/staking/instructions.rs */
043 | pub fn init_config_account(svm: &mut litesvm::LiteSVM, payer: &Keypair, pyth_token_mint: Pubkey) {
048 |     let init_config_data = staking::instruction::InitConfig {
049 |         global_config: GlobalConfig {
054 |             unlocking_duration: UNLOCKING_DURATION,
063 |         },
064 |     };
```

Resolution

This issue has been resolved by PR#508.

Appendix: Methodology and Scope of Work

Assisted by the Sec3 Scanner developed in-house, the manual audit particularly focused on the following work items:

- Check common security issues.
- Check program logic implementation against available design specifications.
- Check poor coding practices and unsafe behavior.
- The soundness of the economics design and algorithm is out of scope of this work

DISCLAIMER

The instance report ("Report") was prepared pursuant to an agreement between Coderrect Inc. d/b/a Sec3 (the "Company") and Pyth Data Association dba Pyth Network (the "Client"). This Report solely includes the results of a technical assessment of a specific build and/or version of the Client's code specified in the Report ("Assessed Code") by the Company. The sole purpose of the Report is to provide the Client with the results of the technical assessment of the Assessed Code. The Report does not apply to any other version and/or build of the Assessed Code. Regardless of the contents of the Report, the Report does not (and should not be interpreted to) provide any warranty, representation or covenant that the Assessed Code: (i) is error and/or bug free, (ii) has no security vulnerabilities, and/or (iii) does not infringe any third-party rights. Moreover, the Report is not, and should not be considered, an endorsement by the Company of the Assessed Code and/or of the Client. Finally, the Report should not be considered investment advice or a recommendation to invest in the Assessed Code and/or the Client.

This Report is considered null and void if the Report (or any portion thereof) is altered in any manner.

ABOUT

The Sec3 audit team comprises a group of computer science professors, researchers, and industry veterans with extensive experience in smart contract security, program analysis, testing, and formal verification. We are also building automated security tools that incorporate static analysis, penetration testing, and formal verification.

At Sec3, we identify and eliminate security vulnerabilities through the most rigorous process and aided by the most advanced analysis tools.

For more information, check out our website and follow us on twitter.