Scope

Audit

Presented by:

OtterSec contact@osec.io

Akash Gurugunti sud0u53r.ak@osec.io

Tamta Topuria tamta@osec.io

Robert Chen r@osec.io

Contents

Appendices

01 | Executive Summary

Overview

Kamino Finance engaged OtterSec to assess the `scope` program. This assessment was conducted between November 28th and December 16th, 2023. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 6 findings throughout this audit engagement.

In particular, we found a vulnerability in a faulty price validation check in the switchboard, resulting in the price being considered valid even if it failed to meet the minimum confirmation requirements specified by the aggregator (OS-SCP-ADV-00). We further highlighted the lack of verification of the exponents' sign during the calculation of prices from Pyth data (OS-SCP-ADV-02) and another issue where the oracle price corresponded to an older price due to a lack of immediate refresh after updating the oracle mapping (OS-SCP-ADV-01).

We also provided suggestions regarding the elimination of unnecessary code in specific functions (OS-SCP-SUG-01) and proposed enhancements to the code base (OS-SCP-SUG-02). Additionally, we recommended modifying the get price method to explicitly verify the status of the price feed and return the value accordingly (OS-SCP-SUG-00).

02 | Scope

The source code was delivered to us in a git repository at github.com/hubbleprotocol/scope. This audit was performed against commit c42e3fd.

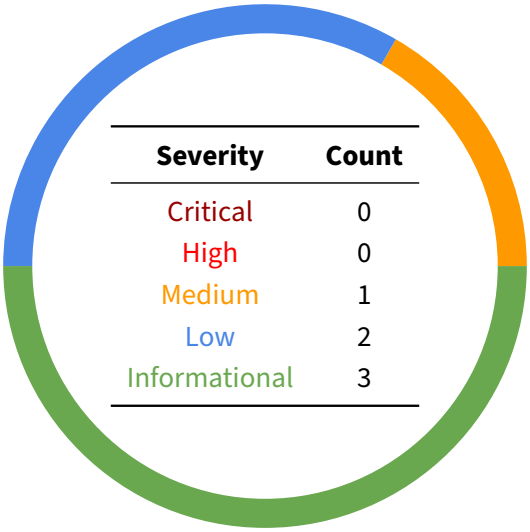
A brief description of the programs is as follows:

Name	Description
scope	A price oracle aggregator for the Solana network which copies data from multiple on-chain oracles accounts into one price feed, pre-validating the prices with a preset of rules and performing the update only if they meet the criteria.

03 | Findings

Overall, we reported 6 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
Critical	0
High	0
Medium	1
Low	2
Informational	3

04 | Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-SCP-ADV-00	Medium	Resolved	`validate_valid_price` uses `min` instead of `max` when calculating `min_num_success_for_oracle`.
OS-SCP-ADV-01	Low	Resolved	Due to a lack of an immediate refresh in `UpdateOracleMapping`, there is a window during which the price in `OraclePrices` corresponds to the old token's price.
OS-SCP-ADV-02	Low	Resolved	Exponents (scale factor) sign is neglected when calculating prices from Pyth data in `pyth` and `pyth_ema`, resulting in improper unit conversions if the exponent is negative.

OS-SCP-ADV-00 [med]| Validation Threshold Mismatch

Description

`validate_valid_price` within `switchboard_v1` calculates the minimum of `aggregator_min_confirmations` and `MIN_NUM_SUCCESS` to determine the minimum number of successful rounds required for the oracle to consider the price valid. This calculation implies that if `MIN_NUM_SUCCESS` is greater than `aggregator_min_confirmations`, it may result in an unintended behavior.

oracles/switchboard_v1.rs RUST

```rust
pub fn validate_valid_price(
    price: u64,
    slot: u64,
    unix_timestamp: u64,
    aggregator: AggregatorState,
    round_result: RoundResult,
) -> Result<DatedPrice> {
    [...]
    let min_num_success_for_oracle = min(aggregator_min_confirmations,
        ↪  MIN_NUM_SUCCESS);
    let num_success = round_result.num_success.ok_or_else(|| {
        msg!("Price not valid: num_success not set");
        ScopeError::PriceNotValid
    })?;
    if num_success < min_num_success_for_oracle {
        msg!("Price not valid: num_success < min_num_success_for_oracle,
            ↪  {num_success} < {min_num_success_for_oracle}",);
        return err!(ScopeError::PriceNotValid);
    };

    Ok(dated_price)
}
```

Specifically, if `round_result.num_success` is greater than or equal to `MIN_NUM_SUCCESS`, the validation would pass regardless of the value of `aggregator_min_confirmations`. This may result in a situation where the program considers the price valid even if it does not fufill the minimum confirmation requirements specified by the aggregator.

Proof of Concept

1. Let `aggregator_min_confirmations` = 5;
2. Let `MIN_NUM_SUCCESS` = 3;
3. Let `num_success` = 4;

Here, `MIN_NUM_SUCCESS` is greater than `aggregator_min_confirmations`, and `num_success` is equal to four. The `min_num_success_for_oracle` calculation results in three, which allows the validation to pass even though the number of successful rounds does not satisfy the aggregator's minimum confirmation requirement.

Remediation

Ensure the calculation of `min_num_success_for_oracl` uses `max` instead of `min`:

oracles/switchboard_v1.rs RUST

```rust
[...]
let min_num_success_for_oracle = max(aggregator_min_confirmations,
    ↪  MIN_NUM_SUCCESS);
[...]
```

This change ensures that `min_num_success_for_oracle` is set to the maximum of the two values, preventing the unintended behavior described above.

Patch

Fixed in PR#221.

OS-SCP-ADV-01 [low] | Inconsistent Oracle Price Update

Description

There is a potential inconsistency in the `OraclePrices` structure when the `UpdateOracleMapping` instruction updates the oracle mappings, particularly when updating the source for an existing token.

Without an immediate refresh, there is a window of time during which the price in `OraclePrices` corresponds to the old token's price and not the new one in `OracleMappings`, resulting in the protocol utilizing the wrong price.

Remediation

Ensure to call `RefreshOne` as a Cross Program Invocation within the `UpdateOracleMapping` instruction. This ensures that the `OraclePrices` structure is always immediately updated with the new token information.

Patch

Fixed in PR#221.

OS-SCP-ADV-02 [low] | Negative Exponent

Description

In `pyth` and `pyth_ema`, while calculating the price from `pyth_price`, the case where the exponent (`expo`) of the price is negative is not handled.

oracles/pyth_ema.rs RUST

```rust
pub fn get_price(price_info: &AccountInfo) -> Result<DatedPrice> {
    [...]
    Ok(DatedPrice {
        price: Price {
            value: price,
            exp: pyth_price.expo.abs().try_into().unwrap(),
        },
        last_updated_slot: price_account.valid_slot,
        unix_timestamp: u64::try_from(price_account.timestamp).unwrap(),
        ..Default::default()
    })
}
```

The code only considers the absolute value of the exponent when calculating the price from Pyth data. The exponent represents the scale factor for the price, determining the number of decimal places to move the implied decimal point. Thus, ignoring the sign of the exponent may result in incorrect unit conversions. In cases where the exponent is negative, neglecting the sign could result in miscalculations of the price units.

Remediation

Handle negative exponents properly in `pyth` and `pyth_ema`, to ensure accurate unit conversions.

Patch

Fixed in PR#221.

05 | General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-SCP-SUG-00	Incorrect price validation check.
OS-SCP-SUG-01	`get_price` and `get_prices_from_data` have redundant code.
OS-SCP-SUG-02	Suggestions regarding code improvements.

OS-SCP-SUG-00 | Refine EMA Price Handling

Description

There is an inconsistency in the price validation check in `get_price`. Currently, `get_ema_price` fails to return None if the EMA price is unavailable. The comment in `get_ema_price` specifies that the method *"currently cannot return None, but may do so in the future"*. This implies that the method always returns Some(`...`) regardless of the availability of the EMA price.

oracles/switchboard_v1.rs RUST

```rust
/// Get the exponential moving average price (ema_price) and a confidence interval
    ↪ on the
/// result.
///
/// Returns `None` if the ema price is currently unavailable.
/// At the moment, the confidence interval returned by this method is computed in
/// a somewhat questionable way, so we do not recommend using it for high-value
    ↪ applications.
pub fn get_ema_price(&self) -> Option<Price> {
    // This method currently cannot return None, but may do so in the future.
    Some(Price {
        price: self.ema_price,
        conf:  self.ema_conf,
        expo:  self.expo,
    })
}
```

Remediation

Modify the method to explicitly check the status of the `PriceFeed` and return None if the status is not `Trading`. This aligns with the intention of skipping the update if the EMA price is currently unavailable.

Patch

Fixed in PR221.

OS-SCP-SUG-01 | Redundant Code

Description

1. The first byte type discriminator check in `get_price` within `switchboard_v1` should be removed and instead be performed while setting price info accounts in the oracle mapping; this ensures better optimization by reducing unnecessary computation during the price retrieval process.

oracles/switchboard_v1.rs RUST

```rust
pub fn get_price(switchboard_feed_info: &AccountInfo) -> Result<DatedPrice> {
    let account_buf = switchboard_feed_info.try_borrow_data()?;
    // first byte type discriminator
    if account_buf[0] != SwitchboardAccountType::TYPE_AGGREGATOR as u8 {
        msg!("switchboard address not of type aggregator");
        return err!(ScopeError::UnexpectedAccount);
    }
    [...]
}
```

2. Within `ktokens` and `ktokens_token_x`, while calling `get_prices_from_data`, the `clmm` parameter is passed to it. However, it is not utilized in fetching reward token prices, and passing it as `None` can avoid unnecessary computations and potentially improve efficiency.

Remediation

Remove the redundant code from `get_price` and `get_prices_from_data`.

Patch

Fixed in PR221.

OS-SCP-SUG-02 | Code Maturity

Description

1. In `refresh_one_price`, the `price_info` account is expected to be associated with a valid oracle mapping in the `oracle_mappings` account, thus, if `price_info.key()` is equal to `Pubkey::default()`, the program should abort as it indicates `price_info` is an uninitialised token.
2. The comment in `refresh_price_list` is incorrect and should be reversed, such that it reads: *// Check the received token list is at least as long as the number of provide*.
3. In `try_from` in `switchboard_v2`, the condition should be `sb_decimal.mantissa <= 0` instead of `sb_decimal.mantissa < 0`. This change ensures that the function will return a `PriceNotValid` error if the mantissa is zero or negative.

oracles/switchboard_v2.rs RUST

```
fn try_from(sb_decimal: SwitchboardDecimal) -> std::result::Result<Self,
    ↪  Self::Error> {
    if sb_decimal.mantissa < 0 {
        msg!("Switchboard v2 oracle price feed is negative");
        return Err(ScopeError::PriceNotValid);
    }
    [...]
}
```

4. In `ktokens`, the program assumes that the price is zero when `shares_issued` is zero. Conversely, the protocol triggers an error in `ktokens_token_x` when `shares_issued` is zero. Ensure this is intended, and if not, modify the behavior when `shares_issued` is zero.

Remediation

Implement the above mentioned suggestions.

Patch

Fixed in PR221.

A | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings section.

Critical	Vulnerabilities that immediately result in a loss of user funds with minimal preconditions. Examples: • Misconfigured authority or access control validation. • Improperly designed economic incentives leading to loss of funds.
High	Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit. Examples: • Loss of funds requiring specific victim interactions. • Exploitation involving high capital requirement with respect to payout.
Medium	Vulnerabilities that may result in denial of service scenarios or degraded usability. Examples: • Computational limit exhaustion through malicious input. • Forced exceptions in the normal user flow.
Low	Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions.
Informational	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants. • Improved input validation.

B | Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.