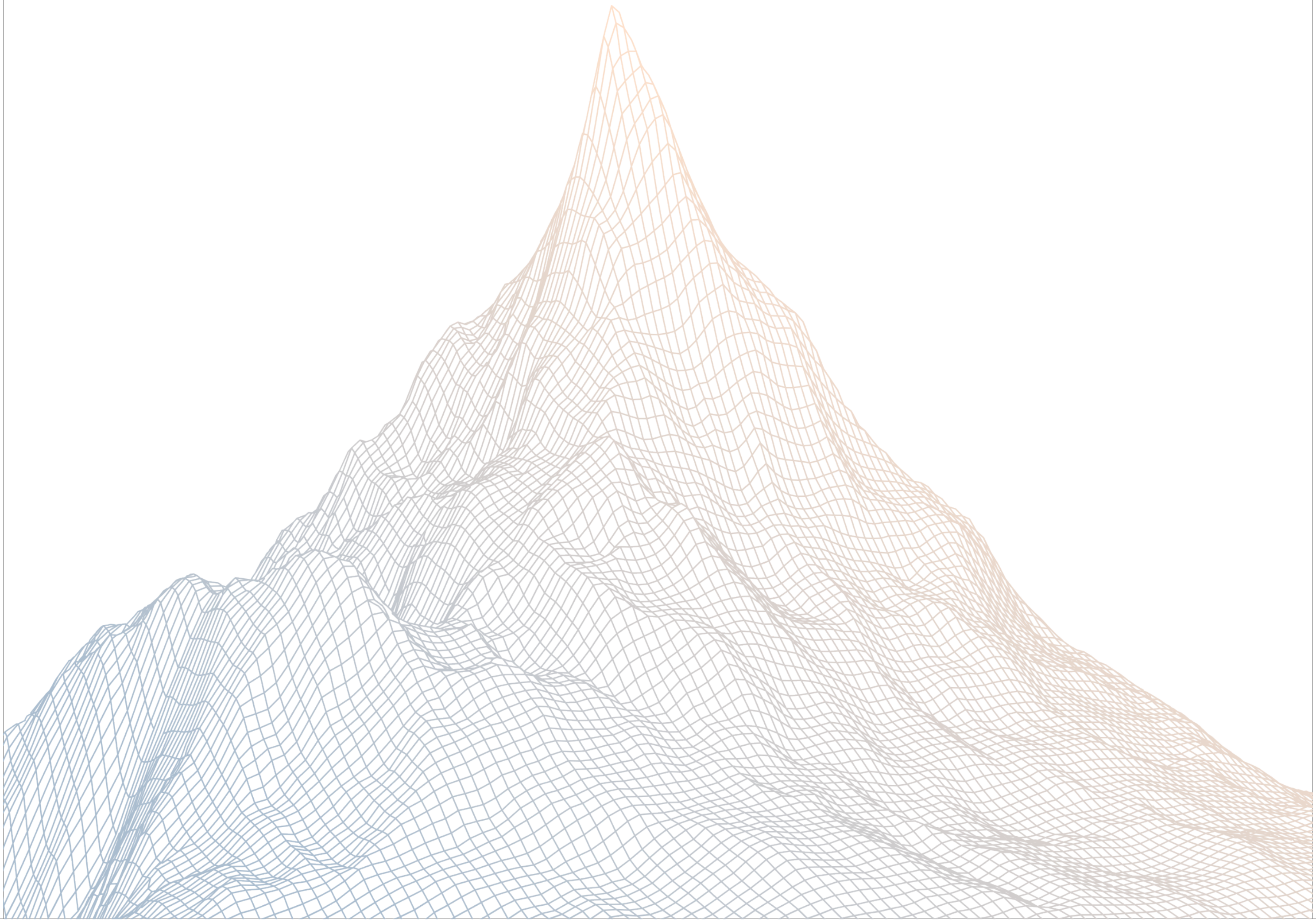
Zenith

RedStone

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES:	October 15th to October 21st, 2025
AUDITED BY:	0x37 peakbolt

Contents

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2 Executive Summary

2.1 About RedStone

RedStone is a data ecosystem that delivers frequently updated, reliable and diverse data for your dApp and smart contracts.

It uses a radically different way of putting data on-chain. The data is automatically attached to a user's transaction and erased afterwards thus reducing gas fees without touching the expensive evm storage.

2.2 Scope

The engagement involved a review of the following targets:

Target	redstone-oracles-monorepo
Repository	https://github.com/redstone-finance/redstone-oracles-monorepo
Commit Hash	`3fd4c2d9a0f5838b785c0eab345d42a760c8ed75`
Files	`packages/sui-connector/sui/contracts/**/sources/**/*.move`

2.3 Audit Timeline

October 15, 2025	Audit start
October 21, 2025	Audit end
October 21, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	1
Medium Risk	2
Low Risk	0
Informational	2
Total Issues	**5**

3 Findings Summary

ID	Description	Status
H-1	try_process_payload() uses unfiltered data packages for aggregated_values	Resolved
M-1	Missing duplicate check for data_points for the same feed_id in extract_values_by_feed_id()	Resolved
M-2	Data package with zero value data point can still be used to meet signer threshold	Resolved
I-1	The malicious signer may impact the median value if the threshold is low	Acknowledged
I-2	Error code for E_DEPRECATED collided with E_INVALID_VERSION	Resolved

4 Findings

4.1 High Risk

A total of 1 high risk findings were identified.

[H-1] `try_process_payload()` uses unfiltered data packages for `aggregated_values`

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: Low

Target

- payload.move#L97-L99

Description:

The function `try_process_payload()` will first parse the raw payload and then filter/verify the `data_packages` based on feed id. It then calculates the median `aggregated_value` from the values in `data_packages`.

However, the `aggregated_value` were calculated based on the initial `parsed_payload` before zero values were filtered out. These zero values are not verified in `try_verify_data_packages()` as they had been filtered out.

This will cause the `aggregated_value` to be incorrect if there are zero values as they are not verified.

```
public fun try_process_payload(
    config: &Config,
    timestamp_now_ms: u64,
    feed_id: vector<u8>,
    payload: vector<u8>,
): Result<ParsedPayload> {
    let parsed_payload = parse_raw_payload(payload);

    let data_packages = parsed_payload.map_ref!(
        |parsed_payload| filter_packages_by_feed_id(
            &data_packages(parsed_payload),
            &feed_id,
        ),
    );

    let data_packages = data_packages.map!(
        |data_packages| filter_out_zero_values(
            data_packages,
        ),
    );

    let verification_result = data_packages.flat_map!(
        |data_packages| try_verify_data_packages(
            &data_packages,
            config,
            timestamp_now_ms,
        ),
    );

    if (!verification_result.is_ok()) {
        return error(verification_result.unwrap_err().into_bytes())
    };

    let values = parsed_payload.map!(
        |parsed_payload| extract_values_by_feed_id(&parsed_payload,
    &feed_id),
    );

    let aggregated_value = values.flat_map!(
        |values| try_calculate_median(
            &mut values.map!(|bytes| from_bytes_to_u256(&bytes)),
        ),
    );

    aggregated_value.map_both!(
        data_packages,
        |aggregated_value, data_packages| ParsedPayload {
            aggregated_value,
            new_package_timestamp: data_packages[0].timestamp(),
        },
    )
}
```

Recommendations:

This can be resolved by using the filtered `data_packages` for calculation of `aggregated_value`.

RedStone: Fixed in @621b518...

Zenith: Verified.

4.2 Medium Risk

A total of 2 medium risk findings were identified.

[M-1] Missing duplicate check for `data_points` for the same `feed_id` in `extract_values_by_feed_id()`

SEVERITY: Medium	IMPACT: High
STATUS: Resolved	LIKELIHOOD: Low

Target

- payload.move#L118-L125

Description:

`extract_values_by_feed_id()` will flatten all the `data_points` in the data packages and filter based on `feed_id` for the median calculation.

However, it does not check that there are no duplicates `data_points` for the same `feed_id`.

This allows a rogue signer to include multiple `data_points` for the same `feed_id` and skew it towards a certain direction and manipulate the median calculation.

```
public fun extract_values_by_feed_id(payload: &Payload, feed_id:
    &vector<u8>): vector<vector<u8>> {
    payload
        .data_packages()
        .map!(|package| *package.data_points())
        .flatten()
        .filter!(|data_point| data_point.feed_id() == feed_id)
        .map!(|data_point| *data_point.value())
}
```

Recommendations:

This can be resolved by checking there is is no duplicate `data_point` for the same `feed_id`.

RedStone: Fixed in @621b518...

Zenith: Verified. Resolved by verifying there are no duplicate `feed _id` in the data packages.

[M-2] Data package with zero value data point can still be used to meet signer threshold

SEVERITY: Medium	IMPACT: High
STATUS: Resolved	LIKELIHOOD: Low

Target

- payload.move#L65-L93

Description:

in `try_process_payload` it will first `filter_packages_by_feed_id` and then `filter_out_zero_values`.

However, `filter_out_zero_values` will retain any packages that had contained a zero value data point for the specified `feed_id`, even after filtering out the zero value data point.

This will cause those particular packages to count towards the signer threshold even though it no longer contain data point for the specified `feed_id`.

Example,

- Suppose we have 3 data packages A1, A2 and A3, that contains data point for the specified feed id A and feed id B.
- And A3 has a zero value data point for feed id A, while A1 and A2 has non-zero data points.
- `filter_packages_by_feed_id(A)` will retain all A1,A2,A3 data packages as they have data point for feed id A.
- However, `filter_out_zero_values` will ony filter out the zero data point for A3, leaving its non-zero data point for feed id B. As A3 still has a non-zero data point for feed id B, it is retained and not entirely filtered out.
- Now A1, A2, A3 will count towards the signer threshold even though A3 no longer has a data point for feed id A.

```
public fun try_process_payload(
    config: &Config,
    timestamp_now_ms: u64,
    feed_id: vector<u8>,
    payload: vector<u8>,
): Result<ParsedPayload> {
    let parsed_payload = parse_raw_payload(payload);

    let data_packages = parsed_payload.map_ref!(
        |parsed_payload| filter_packages_by_feed_id(
            &data_packages(parsed_payload),
            &feed_id,
        ),
    );

    let verification_result = data_packages.flat_map!(
        |data_packages| try_verify_feeds_in_data_packages(
            &data_packages,
        ),
    );
    if (!verification_result.is_ok()) {
        return error(verification_result.unwrap_err().into_bytes())
    };

    let data_packages = data_packages.map!(
        |data_packages| filter_out_zero_values(
            data_packages,
        ),
    );
```

Recommendations:

This can be resolved by performing the `filter_packages_by_feed_id` after `filter_out_zero_values`.

RedStone: Fixed in @3f2f4b8...

Zenith: Verified. Issue is resolved by filtering out data packages with zero value data points, instead of just filtering out data points. This ensures that data packages with zero value datapoints are not used.

4.3 Informational

A total of 2 informational findings were identified.

[I-1] The malicious signer may impact the median value if the threshold is low

SEVERITY: Informational	IMPACT: Medium
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- validate.move

Description:

In median.move, when we fetch different price values from different signers, we will get one median value for this price timestamp. The `signer_count_threshold` will be larger than 0.

If we set the `signer_count_threshold` to 1 or 2, then one malicious signer can manipulate the price.

Suggest that the `signer_count_threshold` should start from 3.

```
public fun try_calculate_median(values: &mut vector<u256>): Result<u256> {
    let len = values.length();

    if (len == 0) {
        return error(b"Empty vector given to median")
    };

    if (len == 1) {
        return ok(values[0])
    };

    if (len == 2) {
        let a = values[0];
        let b = values[1];

        return ok(a / 2 + b / 2 + (a % 2 + b % 2) / 2)
    };
}
fun check(config: &Config) {
    ...
    assert!(config.signer_count_threshold > 0,
    E_SIGNER_COUNT_THRESHOLD_CANT_BE_ZERO);
}
```

Recommendations:

As recommended in description.

RedStone: Acknowledged.

[I-2] Error code for E_DEPRECATED collided with E_INVALID_VERSION

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Informational

Target

- constants.move#L6

Description:

The E_DEPRECATED error uses the same error code as E_INVALID_VERSION. This could cause confusion when an abort occurs.

```
// == Errors ==
const E_DEPRECATED: u64 = 0;
```

Recommendations:

This can be resolved by using a different error code for E_DEPRECATED.

RedStone: Fixed in @621b518...

Zenith: Verified. Resolved by setting E_DEPRECATED = 100.