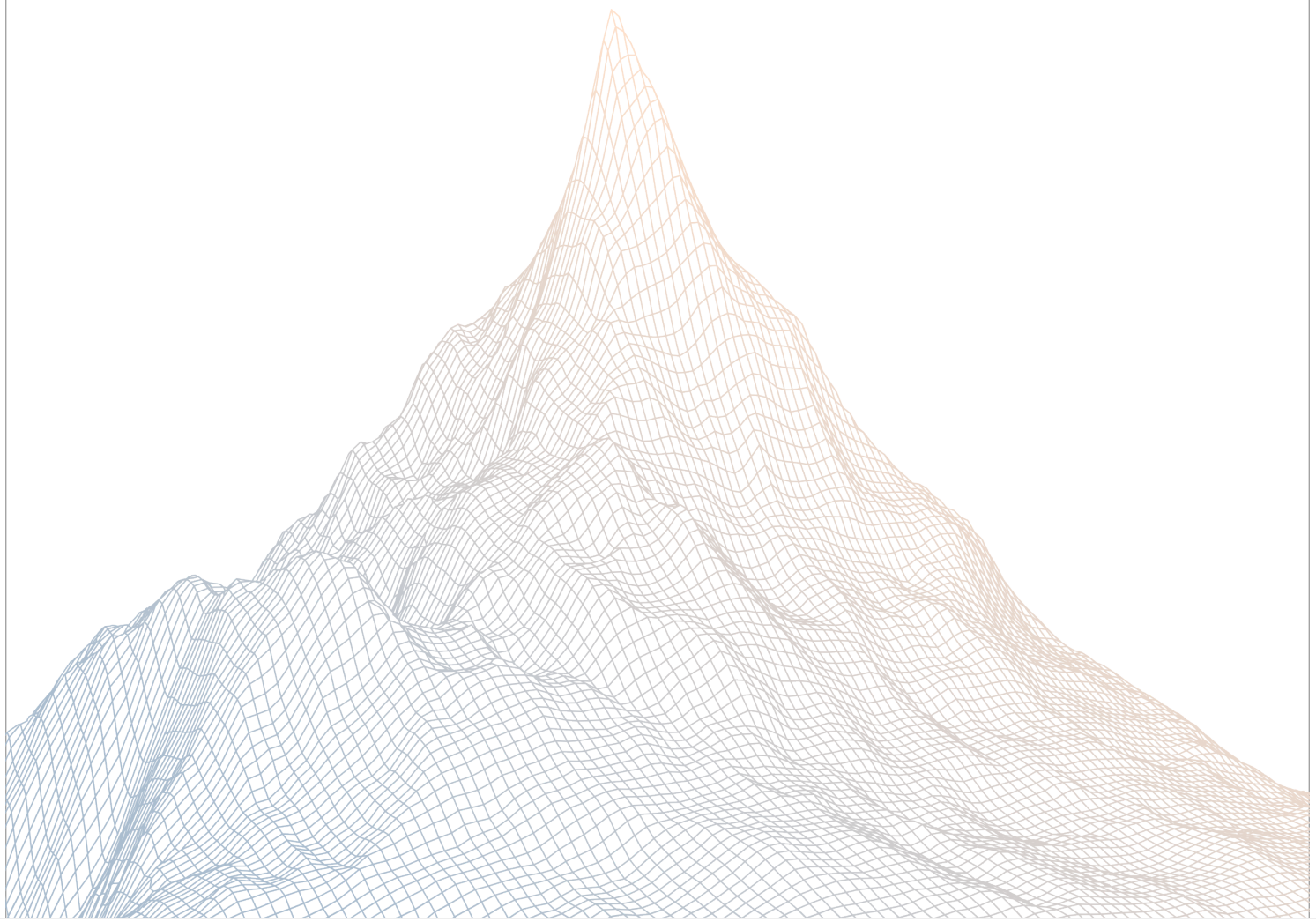
Zenith

RedStone

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: October 2nd to October 7th, 2025

AUDITED BY: Bernd
oakcobalt

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	rust-sdk
Repository	https://github.com/redstone-finance/rust-sdk/
Commit Hash	`de1deeb2d02887fd03e5d295a17ef16342cccfc8`
Files	`mod.rs` `helpers.rs`

Target	redstone-oracles-monorepo
Repository	https://github.com/redstone-finance/redstone-oracles-monorepo
Commit Hash	`2c50510b4368067d32ed0b434432ef23f75b64a4`

Files	`common/src`
	`contracts/redstone-adapter`
	`contracts/redstone-price-feed`

Target	rust-sdk
Repository	https://github.com/redstone-finance/rust-sdk/
Commit Hash	`13abf26c6f938f11b1532d0bd004eb8b8a1dae10`

Files	`crates/redstone/src`
	`crates/redstone/src/core/test_helpers.rs (excluded)`
	`crates/redstone/src/helpers/* (excluded)`
	`Other test code/dependencies (excluded)`

2.3 Audit Timeline

October 2, 2025	Audit start
October 7, 2025	Audit end
October 17, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	5
Low Risk	4
Informational	7
Total Issues	**16**

3

Findings Summary

ID	Description	Status
M-1	write_prices will return prices that failed timestamp validation	Resolved
M-2	recover_address might cause panic and potential DoS of integration flow	Resolved
M-3	SignerAddress::new lowercasing binary data can corrupt the signer address	Resolved
M-4	read_price_data will panic instead of intended error propagation, causing integration flow DOS	Resolved
M-5	Event size limit violation enables DoS via metadata manipulation	Resolved
L-1	trim_end silent return when len overflow, resulting in input malleability	Resolved
L-2	get_prices can be used to return manipulated price data	Acknowledged
L-3	Untrusted updaters cannot fully prevent oracle staleness during trusted updater outage	Acknowledged
L-4	Potential resource exhaustion in current implementation of check_no_duplicates	Resolved
I-1	Unvalidated Payload Metadata Emitted in Events	Resolved
I-2	Unused method	Resolved
I-3	Unchecked data package count	Acknowledged
I-4	Median-of-three optimized implementation reduces readability	Acknowledged
I-5	FeedId size is coupled to VALUE_SIZE instead of DATA_FEED_ID_BS	Acknowledged

ID	Description	Status
I-6	RedStonePriceFeed::description function returns malformed string due to incorrect serialization	Resolved
I-7	Missing zero value validations in feedIds and signerAddresses	Resolved

4

Findings

4.1 Medium Risk

A total of 5 medium risk findings were identified.

[M-1] write_prices will return prices that failed timestamp validation

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

Target

- redstone-adapter/src/lib.rs#L113

Description:

`write_prices` performs two layer of payload validation- (1) `get_prices_from_payload` that checks signature, quorum, datapackage timestamp consistency, etc; (2) `update_feed` checks package timestamp must be strictly increasing and frequency to prevent backward price feed updates;

The issue is `write_prices` will simply return all prices that passed the first validation. This means that prices that failed the second validation step and were not written to storag will also be returned as if they are valid priceData.

```
    pub fn write_prices(
        env: &Env,
        updater: Address,
        feed_ids: Vec<String>,
        payload: Bytes,
    ) → Result<(u64, Vec<(String, U256)>), Error> {
...
        let (package_timestamp, prices) =
            get_prices_from_payload(env, &feed_ids,
    &payload).map_err(error_from_redstone_error)?;
        let write_timestamp = now(env);
...
        let mut updated_feeds = Vec::new(env);

        for (feed_id, price) in prices.iter() {
```

```
            let price_data = PriceData {
                price,
                package_timestamp,
                write_timestamp: write_timestamp.as_millis(),
            };

|>          if update_feed(&db, &verifier, &feed_id, &price_data) {
                updated_feeds.push_back(price_data.clone());
            }
        }

        env.events().publish_event(&WritePrices {
            updated_feeds,
            payload,
            updater,
        });

|>      Ok((package_timestamp, prices)) //@audit prices that didn't write to
    storage due to failed timestamp validation will be returned
    }
```

Impacts: Low This doesn't pollute storage with invalid prices, but does allow these prices to be returned and potentially interpreted as updated prices.

Likelihood: High It happens in correct invocations of write_prices.

Recommendations:

Only return valid prices that pass both validation steps. For example:

```
pub fn write_prices(...) -> Result<(u64, Vec<(String, U256)>), Error> {
    // ... existing code ...
    let mut valid_prices = Vec::new(env);

    for (feed_id, price) in prices.iter() {
        let price_data = PriceData { ... };

        if update_feed(&db, &verifier, &feed_id, &price_data) {
            updated_feeds.push_back(price_data.clone());
            valid_prices.push_back((feed_id.clone(), price.clone()));
        }
    }
    Ok((package_timestamp, valid_prices)) // Return only stored prices
}
```

RedStone: The written values are changed to (). The written values are visible in the Event. Resolved in @426ee25e2f...

Zenith: Verified.

[M-2] recover_address might cause panic and potential DoS of integration flow

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- redstone/src/crypto/mod.rs#L60

Description:

`recover_address` doesn't validate the `recover_byte` before passing it into `recover_public_key`.

```
    fn recover_address<A: AsRef<[u8]>, B: AsRef<[u8]>>(
        &mut self,
        message: A,
        signature: B,
    ) → Result<SignerAddress, CryptoError> {
        let signature = signature.as_ref();
        if signature.len() ≠ SIGNATURE_SIZE_BS {
            return Err(CryptoError::InvalidSignatureLen(signature.len()));
        }
        check_signature_malleability(signature)?;
        let recovery_byte = signature[64]; // 65-byte representation
        let msg_hash = self.keccak256(message);
        let key = self.recover_public_key(
|>          recovery_byte - (if recovery_byte ⩾ 27 { 27 } else { 0 }),
            &signature[..64],
            msg_hash,
        )?;
        let key_hash = self.keccak256(&key.as_ref()[1..]); // skip first
    uncompressed-key byte
        Ok(key_hash.as_ref()[12..].to_vec().into()) // last 20 bytes
    }
```

Depending on the host env and `recover_public_key` implementation, an invalid `recovery_byte` can cause panic/trap that reverts the entire tx. For example, in soroban and radix, the sdk key recovery method doesn't propagate errors. (1) Soroban

```
//crates/redstone/src/soroban/mod.rs
    fn recover_public_key(
        &mut self,
        recovery_byte: u8,
        signature_bytes: impl AsRef<[u8]>,
        message_hash: Self::KeccakOutput,
    ) → Result<Bytes, CryptoError> {
...
         //@audit secp256k1_recover → BytesN<65> . Failure will raise
    HostError, causing VM trap
|>      let public_key = self.env.crypto().secp256k1_recover(
            &message_hash.hash,
            &signature,
            recovery_byte.into(),
        );
...
```

(2) Radix

```
//crates/redstone/src/radix/mod.rs
    fn recover_public_key(
        &mut self,
        recovery_byte: u8,
        signature_bytes: impl AsRef<[u8]>,
        message_hash: Self::KeccakOutput,
    ) → Result<Bytes, CryptoError> {
...
|>      let pk =
    CryptoUtils::secp256k1_ecdsa_verify_and_key_recover_uncompressed(hash,
    signature)
            .0
            .to_vec();
```

Impact: Medium-High `recover_address` is called in `process_payload -> process_values` flow where invalid signature/signer needs to be skipped. This issue will cause an invalid signature to revert the batch.

This might also allow exploits of a malicious/compromised signer passing an invalid signature to DoS write_prices.

Likelihood: Medium An invalid recovery id/signature can happen in normal price reporting operations.

Recommendations:

Consider adding input validations on `recovery_byte` in recover_address before calling `self.recover_public_key`.

RedStone: Resolved with @3111e77213b...

Zenith: Verified.

[M-3] `SignerAddress::new` lowercasing binary data can corrupt the signer address

`SEVERITY:` Medium	`IMPACT:` Medium
`STATUS:` Resolved	`LIKELIHOOD:` Medium

Target

- crates/redstone/src/types/signer_address.rs#L25

Description:

While a hex string representation of an address can be safely lowercased, performing the same operation on its binary representation can lead to data corruption. This is the case in `SignerAddress::new`, which converts the `raw_address`, a byte vector, to lowercase via `to_ascii_lowercase()`.

If any of these bytes fall within the ASCII range for uppercase letters (0x41 to 0x5A), `to_ascii_lowercase` will alter them by adding 0x20, thereby corrupting the address. For example, the byte `0x4C` ('L') would be changed to `0x6c` ('l'). This can lead to issues when integrating the rust-sdk.

The following PoC demonstrates how an address can be mutated:

```
#[cfg(test)]
mod tests {
    use super::*;
    use hex_literal::hex;

    #[test]
    fn lowercasing_binary_changes_address_bytes() {
        // Example address as hex string → binary bytes
        let addr: [u8; 20]
    = hex!("8bb8f32df04c8b654987daaed53d6b6091e3b774");

        println!("addr:   {:?}", addr);

        // Sanity: bytes at indexes 5 and 8 are 0x4c ('L') and 0x49 ('I')
        assert_eq!(addr[5], 0x4c);
        assert_eq!(addr[8], 0x49);
```

```
        // Pad to VALUE_SIZE (32) to satisfy SignerAddress::new
        let mut padded = [0u8; VALUE_SIZE];
        padded[..addr.len()].copy_from_slice(&addr);

        // Constructing SignerAddress triggers to_ascii_lowercase on the
    binary
        let signer = SignerAddress::new(padded);
        let mutated = signer.as_ref();

        println!("signer: {:?}", mutated);

        // Bytes changed due to lowercasing binary data (identity mutation)
        assert_eq!(mutated[5], 0x6c); // 'l'
        assert_eq!(mutated[8], 0x69); // 'i'
        assert_ne!(&mutated[..addr.len()], &addr[..]);
    }
}
```

Test output:

```
addr:   [139, 184, 243, 45, 240, 76, 139, 101, 73, 135, 218, 174, 213, 61,
    107, 96, 145, 227, 183, 116]
signer: [139, 184, 243, 45, 240, 108, 139, 101, 105, 135, 218, 174, 213, 61,
    107, 96, 145, 227, 183, 116, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Recommendations:

Consider removing the call to `.to_ascii_lowercase()` on the binary address bytes within the `SignerAddress::new` implementation.

RedStone: Resolved with @3111e77213b...

Zenith: Verified.

[M-4] read_price_data will panic instead of intended error propagation, causing integration flow DOS

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- redstone-price-feed/src/lib.rs#L103

Description:

The `read_price_data` function in the price-feed contract uses the direct version of the contract client method, which causes the function to panic on application-level errors (such as stale data) instead of properly propagating these errors to the caller.

The code uses the direct version of the contract client method:

```
  pub fn read_price_data(env: &Env) -> Result<PriceData, Error> {
      extend_instance_storage(env);

      let feed_id = Self::feed_id(env)?;
      let client = get_adapter_client(env);

|>    Ok(client.read_price_data_for_feed(&feed_id))
  }
```

The `#[contractclient]` macro generates two versions:

1. Direct version(`read_price_data_for_feed`): Returns `PriceData` directly, panics on any error
2. Try version(`try_read_price_data_for_feed`): Returns `Result<Result<PriceData, Error>, Result<Error, InvokeError>>`, properly propagates errors.

The adapter's `read_price_data_for_feed` returns `Result<PriceData, Error>`, which propagates custom errors such as `MISSING_STORAGE_ENTRY` and `DATA_STALENESS`.

```
pub fn read_price_data_for_feed(env: &Env, feed_id: String) ->
    Result<PriceData, Error> {
```

As a result, any custom error in the client call will panic and not propagate to the outer layer.

Impacts: Medium Integration flows that expect `Result<PriceData, Error>` with error propagation can be DOSsed temporarily due to panic.

Likelihoods: Medium This can happen when custom errors are triggered.

Recommendations:

Consider using `Ok(client.try_read_price_data_for_feed(&feed_id).unwrap()?)`.

RedStone: Resolved with @426ee25e2f...

Zenith: Verified

[M-5] Event size limit violation enables DoS via metadata manipulation

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- redstone-adapter/src/lib.rs#L109

Description:

The `write_prices` function publishes the entire RedStone payload in the `WritePrices` event without size validation. While only trusted updaters can call `write_prices` (due to unrealistic min_interval constraints for untrusted updaters), this protection is insufficient because feedId-specific metadata can cause legitimate transactions to fail when payload exceeds Stellar's event size limit.

```
    pub fn write_prices(
        env: &Env,
        updater: Address,
        feed_ids: Vec<String>,
        payload: Bytes,
    ) → Result<(u64, Vec<(String, U256)>), Error> {
...
        env.events().publish_event(&WritePrices {
            updated_feeds,
|>          payload,
            updater,
        });

        Ok((package_timestamp, prices))
    }
```

Currently, the setllar Event size limit is around 16kb and tx payload size limit is 132kb. In addition, price feed metadata is not validated on-chain and can be of variable length. Consider these cases: (1) 5 signer x 49 datapoints

- metadata (0 bytes)
- Payload size approx.: 404 + (320 × 49) = 16084
- Status: approaching limit

(2) 5 signers × 17 datapoints

- metadata : 10KB
- Payload size approx.: 404 + (320 × 17) + 10240 = 16084
- Status: combined with other fields and overheads, event size limit is exceeded

(3) metadata exploits If metadata can be influenced, even with a few datapoints, metadata pushes payload over 16KB.

Impact: Medium - High These cases can bypass trusted updater protections and cause key write_prices flow DOS.

Likelihood: Medium The event size limit can be reached in normal operations and can also be potentially exploited.

Recommendations:

Before event emitting, consider checking the payload size and optionally emitting a hashed payload.

RedStone: Resolved with @426ee25e2f...

Zenith: Verified.

4.2 Low Risk

A total of 4 low risk findings were identified.

[L-1] trim_end silent return when len overflow, resulting in input malleability

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

Target

- crates/redstone/src/utils/trim.rs#L22

Description:

The `trim_end` function silently accepts truncated input when the requested length exceeds the available data, potentially allowing malformed payloads to bypass validation.

```
impl Trim<Vec<u8>> for Vec<u8> {
    fn trim_end(&mut self, len: usize) -> Self {
        if len ≥ self.len() {
|>            core::mem::take(self)
        } else {
            self.split_off(self.len() - len)
        }
    }
}
```

Impact: When len >= self.len(), the function returns the last bytes (potentially empty) instead of raising an error. This allows truncated payloads to be passed as valid input for further processing in `process_payload` flow.

Recommendations:

If there is no legitimate case for `len>self.len()`, consider returning an error when len overflow.

RedStone: Resolved with @3111e77213b...

Zenith: Verified.

[L-2] get_prices can be used to return manipulated price data

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- redstone-adapter/src/lib.rs#L67

Description:

`get_prices` allows anyone to process payloads and receive validated price data. The returned price data is validated for correctness and timeliness(quorum, timestamp, signature verification). But it still allows the returned price data to be manipulated. For example: (1) Median Manipulation:

- 5 signers report ETH: [1000, 1005, 1010, 1015, 1020] → Median = 1010
- Malicious payload with 3 signers: [1000, 1005, 1010] → Median = 1005
- Result: 0.5% price manipulation

(2) Temporal Attacks:

- At timestamp 5000: `write_prices` stores ETH = 1005
- At timestamp 5005: `get_prices` uses payload from timestamp 4995 (ETH = 1000)
- Result: Returns outdated price despite newer data existing

Both scenarios above satisfy existing checks, but return a different price than the canonical value.

Impact: Low `get_prices` doesn't modify storage and there are other read_price methods for direct on-chain price queries.

Likelihood: Medium `get_prices` doesn't restrict access to trusted updators. It can be vulnerable when it's used in price reporting integration flows.

Recommendations:

Consider increasing transparency by optionally returning newer storage entries (if available) alongside payload-derived prices. Alternatively, if get_prices's use case will not go beyond simulation or debugging, consider adding documentation for security and clarity.

RedStone: Acknowledged. We'll try to expose it more clearly in the docs. `get_prices`

returns only the processed data (pull model, the client sends the payload to be processed and expects the values are consitent with the payload sent) and is orthogonal to the storage.

Zenith: Verified. Resolved with clear documentation of the intended use case of `get_prices`.

[L-3] Untrusted updaters cannot fully prevent oracle staleness during trusted updater outage

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- contracts/redstone-adapter/src/config.rs#L25-L38

Description:

The `redstone-adapter` contract includes a rate-limiting mechanism for price updates from untrusted sources (permissionless mechanism), defined by the `min_interval_between_updates_ms` constant. This constant is currently set to **two days**. This means that any address not present in the hardcoded `TRUSTED_UPDATERS` list is restricted to one price update every **48 hours**.

Concurrently, the `read_prices` function (and similarly also the `read_timestamp`, `read_price_data_for_feed`, and `read_price_data` functions) enforces a data freshness requirement through `DATA_STALENESS`, which is set to **30 hours**. If the stored price data is older than this threshold, the function will error, causing all price lookups to fail.

This combination of a long rate-limit for untrusted updaters and a shorter data staleness period creates a liveness risk. Should the four trusted updaters experience downtime or go offline for more than 30 hours, the oracle's price data will become stale and the contract will become unusable. The broader community would be powerless to remedy this situation, as they would be locked out from providing fresh price data for 48 hours after a single update. This configuration centralizes the oracle's liveness around a small, fixed set of trusted updaters, creating a single point of failure.

However, we are aware that this is a very unlikely scenario, therefore, we classify this issue as low-severity.

Recommendations:

It is recommended to reduce the value of `min_interval_between_updates_ms` in `contracts/redstone-adapter/src/config.rs` to a more reasonable value that would allow the wider community to act as a reliable fallback, ensuring the oracle's continued operation if the primary trusted updaters become unavailable, without creating a vector for spam

attacks.

RedStone: The min_interval_between_updates_ms is set to 48 hours, which effectively means infinity for us. However, the trusted updaters are still updating the values, and we expect that 24 hours is the maximum interval for a feed to be present in the RedStone system. This value could probably be decreased to 24 hours + 5 minutes instead of 48 hours, but the current value was chosen to align with the EVM adapters. Therefore, we acknowledge this issue.

[L-4] Potential resource exhaustion in current implementation of `check_no_duplicates`

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

Target

- crates/redstone/src/utils/slice.rs#L26-L27

Description:

The `check_no_duplicates` method currently uses a quadratic algorithm that can be exploited when processing a large number of elements.

```
pub fn check_no_duplicates<T>(slice: &[T]) → Result<(), T>
where
    T: PartialEq + Eq + Copy,
{
    // ... edge cases ...
    for (i, a) in slice.iter().enumerate() {
        for b in slice.iter().skip(i + 1) {  // O(n²) complexity
            if a == b {
                return Err(*a);
            }
        }
    }
    Ok(())
}
```

In current use cases, `check_no_duplicates` is used in input validations for signer address and feed ids. Although signer addresses are pre-configured and capped at 255, feedIds are not capped and passed as input arguments in Oracle adapters.

1. Signer validation: `verify_signers_config()` - limited to 255 signers (u8::MAX)
2. Feed ID validation: `verify_feed_id_list()` - no size limits found

On-chain usage has resource constraints that likely prevent exploits, but off-chain usage has no economic disincentives, which likely cause resource exhaustion.

Recommendations:

Consider using a hyprid approach. e.g.: when slice.len() <= 50, existing quardratic approach works fine and consider using sorted approach for large inputs.

RedStone: Resolved with @3111e77213...

Zenith: Verified. Upper bounds for feedIds added.

4.3 Informational

A total of 7 informational findings were identified.

[I-1] Unvalidated Payload Metadata Emitted in Events

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

Target

- redstone-adapter/src/lib.rs#L109

Description:

Payload metadata is unsigned and not validated, which can be a source of exploits. Currently, `write_prices` will emit `payload` including metadata.

```
    pub fn write_prices(
        env: &Env,
        updater: Address,
        feed_ids: Vec<String>,
        payload: Bytes,
    ) → Result<(u64, Vec<(String, U256)>), Error> {
...
        env.events().publish_event(&WritePrices {
            updated_feeds,
|>          payload,
            updater,
        });

        Ok((package_timestamp, prices))
    }
```

If downstream systems process the event payload and rely on the unsigned metadata for business logic, they may encounter data integrity issues from unvalidated metadata.

Recommendations:

Document that unsigned metadata is not validated, or consider only emitting validated portions of the payload in events.

RedStone: Resolved with @426ee25e2fb...

Zenith: Verified.

[I-2] Unused method

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

Target

- crates/redstone/src/core/validator.rs#L971

Description:

`fn validate_signer_count_threshold` is currently unused. Actual signer count threshold validation in process payload flow doesn't use this method.

```
#[inline]
fn validate_signer_count_threshold(
    &self,
    index: usize,
    values: &[Option<Value>],
) -> Result<Vec<Value>, Error> {
    let values = values.filter_some();

    if values.len() < self.signer_count_threshold() as usize {
        return Err(Error::InsufficientSignerCount(
            index,
            values.len(),
            self.feed_ids()[index],
        ));
    }

    Ok(values)
}
```

Recommendations:

If `fn validate_signer_count_threshold` is not intended to be used elsewhere, consider removing dead code.

RedStone: Resolved with @3111e77213...

Zenith: Verified.

[I-3] Unchecked data package count

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

Target

- crates/redstone/src/protocol/payload_decoder.rs#L60-L73

Description:

The `PayloadDecoder::trim_metadata` function decodes the `data_package_count` from the bytes payload. This value is then used in the `trim_data_packages` function to allocate a `Vec` with a capacity equal to `data_package_count` followed by iterating `count` times.

```
60:  fn trim_data_packages(
61:      &mut self,
62:      payload: &mut Vec<u8>,
63:      count: usize,
64:  ) -> Result<Vec<DataPackage>, Error> {
65:      let mut data_packages = Vec::with_capacity(count);
66:
67:      for _ in 0..count {
68:          let data_package = self.trim_data_package(payload)?;
69:          data_packages.push(data_package);
70:      }
71:
72:      Ok(data_packages)
73:  }
```

While there is a `DATA_POINT_COUNT_MAX_VALUE` constant that caps the number of data points within a package, a similar safeguard is missing for the number of data packages itself.

Recommendations:

Consider introducing a limit for `data_package_count` after the value is read from the payload.

RedStone: Acknowledged.

[I-4] Median-of-three optimized implementation reduces readability

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

Target

- crates/redstone/src/utils/median.rs#L48-L51

Description:

The implementation for finding the median of three elements in `Median::median()` is unnecessarily tricky to understand. The current logic uses a series of nested function calls and `unwrap_or_else`, which obscures the straightforward goal of selecting the middle value.

```
45:  let median = match len {
46:      1 ⇒ self[0],
47:      2 ⇒ self[0].avg(self[1]),
48:      3 ⇒ maybe_pick_median(self[0], self[1], self[2]).unwrap_or_else(|| {
49:          maybe_pick_median(self[1], self[0], self[2])
50:              .unwrap_or_else(|| maybe_pick_median(self[1], self[2],
    self[0]).unwrap())
51:      }),
```

While this implementation may be a clever attempt to avoid sorting, it sacrifices clarity and maintainability. For a small, fixed-size array of three elements, the performance benefits of avoiding a sort are negligible, as the compiler is highly effective at optimizing such operations.

Recommendations:

Consider replacing the complex logic with a simpler, more idiomatic approach that is easier to read and verify. Sorting a three-element array is computationally trivial and makes the code's intent immediately clear.

A clearer implementation would be:

```
let mut v = [self[0], self[1], self[2];
```

```
v.sort_unstable();
v[1]
```

RedStone: Acknowledged.

[I-5] `FeedId` size is coupled to `VALUE_SIZE` instead of `DATA_FEED_ID_BS`

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

Target

- crates/redstone/src/types/feed_id.rs#L13

Description:

The `FeedId`, which represents a 32-byte data feed identifier, is using the `VALUE_SIZE = 32` constant as the length. This constant is intended for the size of a data point's value.

A more semantically correct constant, `DATA_FEED_ID_BS`, exists and is explicitly designated for the size of a data feed ID.

While both constants are currently equal to 32, using `VALUE_SIZE` for `FeedId` creates a fragile and incorrect semantic link between the size of a feed identifier and the size of a data value. A future change to `VALUE_SIZE` would unintentionally alter the size of `FeedId`, likely breaking protocol compatibility.

Recommendations:

Consider changing the definition of `FeedId` to use the `DATA_FEED_ID_BS` constant. This will make the implementation more robust and align the code with the intended semantics of the protocol constants.

RedStone: Acknowledged

[I-6] `RedStonePriceFeed::description` function returns malformed string due to incorrect serialization

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

Target

- contracts/redstone-price-feed/src/lib.rs#L71

Description:

The `RedStonePriceFeed::description` function is intended to return a human-readable string identifying the price feed, for example, "RedStone Price Feed for BTC".

Currently, the implementation constructs this string by appending the `feed_id` to a prefix. However, it uses `feed_id.to_xdr(env)` for this concatenation:

```
65: pub fn description(env: &Env) -> Result<String, Error> {
66:     let feed_id = Self::feed_id(env)?;
67:
68:     let mut description_bytes = Bytes::new(env);
69:
70:     description_bytes.extend_from_array(DESCRIPTION_PREFIX);
71:     description_bytes.append(&feed_id.to_xdr(env));
72:
73:     Ok(String::from_bytes(env, &description_bytes.to_alloc_vec()))
74: }
```

The `to_xdr` method serializes the `String` object into its XDR representation, which includes metadata such as the string's length, rather than just its raw UTF-8 bytes. When these XDR bytes are appended to the prefix and the resulting bytes are converted back to a `String`, the output is malformed because it contains the extra serialization artifacts.

Recommendations:

To ensure the description string is correctly formed, the raw UTF-8 bytes of the `feed_id` should be appended. Consider replacing `feed_id.to_xdr(env)` with `feed_id.to_bytes()`.

The corrected implementation would look like this:

```
description_bytes.append(&feed_id.to_bytes());
```

This change will ensure that only the string content is appended, resulting in the expected descriptive output.

RedStone: Resolved with @e78b8415759...

Zenith: Verified.

[I-7] Missing zero value validations in feedIds and signerAddresses

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

Target

- crates/redstone/src/contract/verification.rs#L154
- crates/redstone/src/core/config.rs#L95-L97

Description:

The RedStone SDK lacks proper validation for zero values in critical configuration parameters (`SignerAddress` and `FeedId`), allowing invalid configurations to pass verification.

Impacts: This allows bypasses where zero-value signers are counted towards signer thresholds. The configuration appears to meet security requirements but lacks actual signers

Recommendations:

Consider adding zero-value check for both signer addresses and feed ids.

RedStone: Resolved with @3111e77213b...

Zenith: Verified.