Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

RedStone

RedStone Stellar Connector

Veridise Inc.
October 06, 2025

► **Prepared For:**

RedStone
https://www.redstone.finance/

► **Prepared By:**

Alberto Gonzalez
Aayushman Thapa Magar

► **Contact Us:**

contact@veridise.com

► **Version History:**

Oct. 27, 2025	V2
Oct. 20, 2025	V1

Contents

1 Executive Summary

From Oct. 6, 2025 to Oct. 14, 2025, RedStone engaged Veridise to conduct a security assessment of their RedStone Stellar Connector. The security assessment covered the Rust SDK, a collection of utilities designed to facilitate the deserialization and verification of the RedStone payload, as well as the Soroban smart contracts, which implement the oracle price-push model for third-party contracts to consume data from. Veridise conducted the assessment over 2 person-weeks, with 2 security analysts reviewing the project over 1 week on commits `ff1de9` (Rust-SDK) [*] and `726ac4` (Soroban contracts) [†]. The review strategy involved a tool-assisted analysis of the program source code performed by Veridise security analysts as well as thorough code review.

Project Summary. The security assessment focused on the RedStone Rust SDK and, the `RedStonePriceFeed` and `RedStoneAdapter` Soroban smart contracts. RedStone is a modular, multi-chain oracle network that delivers market data to smart contracts with a focus on cost efficiency and speed. The system separates data production from on-chain delivery and supports two complementary models; the pull model that includes signed data packages (containing price information) within the user transaction and, the push model that periodically updates on-chain adapters with price information.

The following were the main components for the audit:

- **Rust SDK**. The RedStone Rust SDK is the payload-processing and verification engine. It provides utilities to parse payload bytes into data packages, recover and validate signer identities against a configured registry, enforce timestamp freshness, and aggregate per-feed values across unique signers. It is designed to be compatible with different chain contexts while preserving the same verification and aggregation capabilities.
- **RedStoneAdapter**. This contract serves as an oracle gateway on Stellar that supports two ways to consume prices. First, a per-transaction (pull) path where callers include a RedStone payload and receive a fresh, verified price. Second, a cached (push) path, where a trusted updater records verified prices on-chain so other contracts can read the latest available value without carrying a payload. The RedStoneAdapter contract utilizes the Rust SDK for payload processing and verification.
- **RedStonePriceFeed**. This contract serves as a wrapper around the RedStoneAdapter contract, which provides price information on a per feed basis.

Code Assessment. The RedStone Stellar Connector developers provided the source code of the Rust SDK and Soroban contracts for the code review. The source code appears to be mostly original code written by the RedStone Stellar Connector developers. The repositories contained documentation in the form of READMEs that were helpful in understanding the intended behavior of the codebase. Veridise security analysts' understanding of the project was also furthered by the additional protocol documentation which gave a high-level overview of the RedStone Stellar Connector.

The source code contained a test suite, which the Veridise security analysts noted were extensive and thorough.

[*] `https://github.com/redstone-finance/rust-sdk/tree/ff1de9da45e874ec09366ce4406e110131056a59`

[†] `https://github.com/redstone-finance/redstone-oracles-monorepo/tree/726ac4bd7770ef16ab987d361b4683613bd983a0`

Summary of Issues Detected. The security assessment uncovered 5 issues, with 2 low-severity issues, 2 warnings, and 1 informational finding. Specifically, V-RED-VUL-001 describes that raw addresses were normalized to lowercase as if they were ASCII bytes, which could have resulted in the recovery of unintended addresses. V-RED-VUL-002 highlights that including a null byte in the middle of a feed ID could have allowed malicious users to spoof legitimate feed IDs. Finally, V-RED-VUL-003 emphasizes the importance of implementing a two-step ownership transfer pattern to make the project more robust against human error.

Some findings were determined to reflect intended behavior or were deemed invalid after discussions with the developers; these are detailed in Appendix A.

Recommendations. After conducting the assessment of the protocol, the security analysts had a few suggestions to improve the RedStone Stellar Connector project. Overall, the codebase demonstrates high quality with a comprehensive test suite, and the identified issues were self-contained to specific lines of code requiring only small changes to fix with the issues originating from edge cases.

Incident Response. The Adapter contract operates under a multisignature configuration that requires approval from three out of five trusted signers to authorize price submissions. As discussed in (V-RED-APP-VUL-001), the compromise of only two signers, less than half of the total, could enable the post of arbitrary prices. In light of this risk, the protocol team should plan and adopt a response procedure aimed at mitigating the operational risks stemming from signer compromise. This plan should include a defined process for immediate signer rotation (V-RED-APP-VUL-003), allowing compromised accounts to be replaced quickly through the upgrade mechanisms available in the contracts. The response plan must also account for how the protocol communicates incidents to its downstream integrators and consumers (V-RED-APP-VUL-002). In situations where signer compromise or suspicious price behavior is detected, there should be a clear and timely mechanism to notify third-party consumers not to rely on the current price data until verification or recovery procedures have been completed.

Volatility Scenarios. During periods of high market volatility or network congestion, trusted signers may observe significantly different off-chain price data depending on their data sources or timing of price retrieval. As the Adapter contract aggregates these submissions using a median-based approach, substantial divergence among signer inputs can result in an inaccurate on-chain median that does not reflect the true market value. Since the current implementation does not include any on-chain validation or rejection mechanism for outlier prices, the protocol should implement an off-chain consistency check to mitigate this risk. Establishing such an off-chain validation layer would reduce the likelihood of erroneous median prices being propagated on-chain during extreme volatility.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Soroban Smart Contracts	`726ac4`	Rust	Stellar
Rust SDK	`ff1de9`	Rust	Library

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Oct. 6–Oct. 14, 2025	Manual & Tools	2	2 person-weeks

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	0	0	0
High-Severity Issues	0	0	0
Medium-Severity Issues	0	0	0
Low-Severity Issues	2	2	2
Warning-Severity Issues	2	2	2
Informational-Severity Issues	1	1	0
TOTAL	5	5	4

Table 2.4: Category Breakdown.

Name	Number
Logic Error	2
Usability Issue	1
Data Validation	1
Maintainability	1

3 Security Assessment Goals and Scope

3.1 Security Assessment Goals

The engagement was scoped to provide a security assessment of RedStone Stellar Connector's During the assessment, the security analysts aimed to answer questions such as:

- Are there any language specific flaws or counter patterns present in the code base?
- Does the codebase have common vulnerabilities in Soroban contracts such as: missing authorization checks, improper storage TTL management and excessive resource consumption?
- Are there any flaws that can lead to the contracts reporting wrong price information?
- Are there safeguards against publishing stale data that appears fresh due to timestamp issues?
- Is there a secure mechanism to rotate trusted signers in case of key compromise without requiring a full contract upgrade?
- Is payload processing, including decoding, price aggregation/calculation, timestamp, and signature verification correct?
- Can malicious updaters submit valid signatures for incorrect feed IDs to swap price data between feeds?
- Can trusted updaters be impersonated or their authorization bypassed through signature verification flaws?
- Can an attacker exploit the signer threshold mechanism to inject fraudulent price data?
- Is there a way to prevent legitimate users from interacting with the protocol?

3.2 Security Assessment Methodology

The Security Assessment process consists of the following steps:

- Gather an initial understanding of the protocol's logic, users, and workflows by reviewing the provided documentation and consulting with the developers.
- Identify and prioritize the most significant security risks based on that understanding.
- Systematically review the codebase for execution paths that could trigger the identified security risks, considering different assumptions.
- Prioritize one finding over another by assigning a severity level to each.

3.3 Scope

The scope of this security assessment is limited to a specific set of source files in each repository, as agreed upon with the RedStone Stellar Connector developers:

Rust SDK on commit `ff1de9`:

- `crates/redstone/src/lib.rs`
- `crates/redstone/src/casper/*.rs`
- `crates/redstone/src/contract/*.rs`
- `crates/redstone/src/core/*.rs`
- `crates/redstone/src/crypto/*.rs`
- `crates/redstone/src/default_ext/*.rs`
- `crates/redstone/src/network/*.rs`
- `crates/redstone/src/protocol/*.rs`
- `crates/redstone/src/radix/*.rs`
- `crates/redstone/src/solana/*.rs`
- `crates/redstone/src/soroban/*.rs`
- `crates/redstone/src/types/*.rs`
- `crates/redstone/src/utils/*.rs`

Excluding:

- `crates/redstone/src/core/test_helpers.rs`
- `crates/redstone/src/helpers`

Stellar Contracts on commit `726ac4`:

- `stellarMultiFeed/common/src/lib.rs`
- `stellarMultiFeed/common/src/ownable.rs`
- `stellarMultiFeed/common/src/upgradable.rs`
- `stellarMultiFeed/contracts/redstone-adapter/src/config.rs`
- `stellarMultiFeed/contracts/redstone-adapter/src/event.rs`
- `stellarMultiFeed/contracts/redstone-adapter/src/lib.rs`
- `stellarMultiFeed/contracts/redstone-adapter/src/util.rs`
- `stellarMultiFeed/contracts/redstone-price-feed/src/config.rs`
- `stellarMultiFeed/contracts/redstone-price-feed/src/lib.rs`

Methodology. Veridise security analysts reviewed the reports of previous audits for RedStone Stellar Connector, inspected the provided tests, and read the RedStone Stellar Connector documentation. They then began a review of the code assisted by static analyzers and code vulnerability scanners.

During the security assessment, the Veridise security analysts regularly contacted the RedStone Stellar Connector developers asynchronously to ask questions about the code.

3.4 Classification of Vulnerabilities

When Veridise security analysts discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise.

The severity of a vulnerability is evaluated according to the Table 3.1.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

The likelihood of a vulnerability is evaluated according to the Table 3.2.

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

The impact of a vulnerability is evaluated according to the Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Trust Model

4.1 Operational Assumptions

In addition to assuming that any out-of-scope components behave correctly, Veridise analysts assumed the following properties held when modeling security for RedStone Stellar Connector.

- **Owners and updaters**. It is assumed that high-privilege roles such as owners and updaters do not act maliciously and only perform actions and duties that are expected.
- **Upgrade process**. The Stellar contracts can be upgraded by replacing their WASM. The steps to install the WASM into the ledger is not included in the source code and is assumed to be performed correctly.
- **Out of scope dependencies**. The review of the Rust SDK was limited to the `/crates/redstone/src` path of the repository. Any out-of-scope components are assumed to perform correctly and as intended.
- **Data collection layer**. During the review, the data collection and payload generation components of RedStone are assumed to function correctly and that the price information is collected and signed reliably.

4.2 Privileged Roles

Roles. This section describes in detail the specific roles present in the system, and the actions each role is trusted to perform. The roles are grouped based on two characteristics: privilege-level and time-sensitivity. *Highly-privileged* roles may have a critical impact on the protocol if compromised, while *limited-authority* roles have a negative, but manageable impact if compromised. Time-sensitive *emergency* roles may be required to perform actions quickly based on real-time monitoring, while *non-emergency* roles perform actions like deployments and configurations which can be planned several hours or days in advance.

During the review, Veridise analysts assumed that the role operators perform their responsibilities as intended. Protocol exploits relying on the below roles acting outside of their privileged scope are considered outside of scope.

- Highly-privileged, emergency roles:
  - `RedStoneAdapter.owner` and `RedStonePriceFeed.owner` can change ownership and upgrade the contract.
- Limited-authority, non-emergency roles:
  - `RedStoneAdapter.updater` can update the price information stored in the contract state only if they have valid signatures from the trusted signer set.

Operational Recommendations. Highly-privileged, non-emergency operations should be operated by a multi-sig contract or decentralized governance system. These operations should be guarded by a timelock to ensure there is enough time for incident response. Highly-privileged, emergency operations should be tested in example scenarios to ensure the role operators are available and ready to respond when necessary.

Full validation of operational security practices is beyond the scope of this review. Users of the protocol should ensure they are confident that the operators of privileged keys are following best practices such as:

- Never storing a protocol key in plaintext, on a regularly used phone, laptop, or device, or relying on a custom solution for key management.
- Using separate keys for each separate function.
- Storing multi-sig keys in a diverse set of key management software/hardware services and geographic locations.
- Enabling 2FA for key management accounts. SMS should *not* be used for 2FA, nor should any account which uses SMS for 2FA. Authentication apps or hardware are preferred.
- Validating that no party has control over multiple multi-sig keys.
- Performing regularly scheduled key rotations for high-frequency operations.
- Securely storing physical, non-digital backups for critical keys.
- Actively monitoring for unexpected invocation of critical operations and/or deployed attack contracts.
- Regularly drilling responses to situations requiring emergency response such as pausing/unpausing.

5 Vulnerability Report

This section presents the vulnerabilities found during the security assessment. For each issue found, the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.) is specified. Table 5.1 summarizes the issues discovered:

Table 5.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-RED-VUL-001	Address normalization via . . .	Low	Fixed
V-RED-VUL-002	Null byte injection enables feed ID spoofing	Low	Fixed
V-RED-VUL-003	Two-step ownership transfer pattern not . . .	Warning	Fixed
V-RED-VUL-004	Missing ECDSA signature parameter . . .	Warning	Fixed
V-RED-VUL-005	Informational: Code Quality and Best . . .	Info	Acknowledged

5.1 Detailed Description of Issues

5.1.1 V-RED-VUL-001: Address normalization via to_ascii_lowercase() allows unauthorized signature validation

Severity	Low	**Commit**	ff1de9
Type	Logic Error	**Status**	Fixed
Location(s)	rust-sdk/crates/redstone/src/types/signer_address.rs:25		
Confirmed Fix At	https://github.com/redstone-finance/rust-sdk/pull/60, e24a58a		

Description The RedStone oracle system validates data package signatures by recovering the signer's address from ECDSA signatures and checking if that address belongs to the authorized signer list configured in `Config`. The signature recovery process in `Crypto::recover_address()` recovers the public key from a signature, hashes it with Keccak256, and extracts the last 20 bytes to form an Ethereum-style address. This raw 20-byte address is then wrapped in a `SignerAddress` type via `SignerAddress::from(Vec<u8>)`.

The vulnerability exists in `SignerAddress::new()` at `signer_address.rs`. This function incorrectly applies `to_ascii_lowercase()` to the raw byte array representing the numeric address value. The `to_ascii_lowercase()` method treats any byte in the ASCII uppercase letter range (0x41-0x5A, corresponding to 'A'-'Z') as an ASCII character and transforms it to the corresponding lowercase value by adding 32 (0x20). For example, byte 0x41 becomes 0x61, byte 0x42 becomes 0x62, and so on.

Since an Ethereum address is a 20-byte numeric value, not an ASCII-encoded string, applying ASCII case normalization corrupts the address data. When the recovered address contains any bytes in the range 0x41-0x5A, these bytes are modified, producing a different address than what was actually recovered from the signature.

Impact If an attacker controls a private key whose address matches the corrupted form of a legitimate signer's address, they can create valid signatures that pass validation. Consider a legitimate signer with address `0x4142434445464748494A4B4C4D4E4F5051525354` (containing bytes 'A', 'B', 'C', etc. in the 0x41-0x5A range). After corruption via `to_ascii_lowercase()`, this becomes `0x6162636465666768696a6b6c6d6e6f7071727374`. An attacker controlling the private key for address `0x6162636465666768696a6b6c6d6e6f7071727374` can sign data packages, and the system will incorrectly validate these signatures as coming from the legitimate signer.

This vulnerability can be confirmed with a test demonstrating that two distinct addresses become equal after `SignerAddress` conversion. The test shows addresses `0x8bb8f32df04c8b654987daaed53d6b6091e3b774` and `0x8bb8f32df06c8b656987daaed53d6b6091e3b774` collapse to the same `SignerAddress` value, despite being cryptographically distinct addresses controlled by different private keys. The following test can be copied and pasted in the `config.rs` file at `StellarMultiFeed/contracts/redstone-adapter`.

```
#[test]
fn test_signer_address_to_ascii_lowercase_issue() {

    let legitimate_signer_raw_address: [u8; 20] = hex!("8bb8f32df04c8b654987daaed53d6b6091e3b774");

    let malicious_signer_raw_address: [u8; 20] =  hex!("8bb8f32df06c8b656987daaed53d6b6091e3b774");

    // @audit Not equal raw_addresses.
    assert_ne!(legitimate_signer_raw_address, malicious_signer_raw_address);

    let legitimate_signer_address_converted: SignerAddress = legitimate_signer_raw_address.to_vec().into();

    let malicious_signer_address_converted: SignerAddress = malicious_signer_raw_address.to_vec().into();

    // @audit Equal SignerAddresses.
    assert_eq!(legitimate_signer_address_converted, malicious_signer_address_converted);
}
```

Recommendation Remove the `to_ascii_lowercase()` call from `SignerAddress::new()`. After removing this transformation, verify that the existing test suite passes without requiring any test data modifications. If tests fail due to the removal, investigate whether those tests were written to accommodate the incorrect behavior rather than validate correct functionality. Additionally, add the vulnerability demonstration test to the test suite and ensure it correctly fails the final assertion (expecting the two distinct addresses to remain unequal after conversion).

Developers Response The developers implemented the suggested recommendation.

5.1.2 V-RED-VUL-002: Null byte injection enables feed ID spoofing

Severity	Low	**Commit**	`726ac4`
Type	Logic Error	**Status**	Fixed
Location(s)	`redstone-oracles-monorepo/packages/[...]/utils.rs:9`		
Confirmed Fix At	`https://github.com/redstone-finance/redstone-oracles-monorepo/commit/e78b8415759bb7d8d0b31972051836451016a729`, e78b841		

Description The RedStone adapter contract processes oracle price data in `get_prices_from_payload()` by calling the RedStone library's `process_payload()`function. This function returns validated price data containing `FeedValue` structs, each with a `feed` field of type `FeedId`. These `FeedId` values represent the feed identifiers that were signed by the oracle signers and verified on-chain. The `FeedId` type is a wrapper around a 32-byte array (`[u8; 32]`). After `process_payload()` returns, the contract must convert these `FeedId` values back to `String` format for storage, using the `feed_to_string()` function from the `utils.rs` file.

The `feed_to_string()` function converts a `FeedId` to `String` by splitting the byte array at the first null byte (0x00) and converting only the prefix. This implementation assumes feed identifiers contain no null bytes in the middle of valid data. However, if oracle signers sign a `FeedId` containing embedded null bytes (e.g., `[66, 84, 67, 0, 67, ...]` representing "BTC[null]C"), the `feed_to_string()`function truncates at the first null and returns "BTC". This is demonstrated by pasting and running the following test in the `redstone-adapter/src/utils.rs` file, where two distinct `FeedId` values produce identical strings:

```rust
#[test]
fn test_feed_to_string_simple() {
        let env = Env::default();

        let btc_feed_id_array: [u8; 32] = [
            66, 84, 67, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0,
        ];
        let btc_feed_id = FeedId::from(btc_feed_id_array);
        let convert_btc_feed_id_to_string = feed_to_string(&env, btc_feed_id);

        let non_btc_feed_id_array: [u8; 32] = [
            66, 84, 67, 0, 67, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0,
        ];
        let non_btc_feed_id = FeedId::from(non_btc_feed_id_array);
        let convert_non_btc_feed_id_to_string = feed_to_string(&env, non_btc_feed_id);

        assert_eq!(
            convert_btc_feed_id_to_string,
            convert_non_btc_feed_id_to_string
        );
}
```

Impact An attacker can exploit this by convincing oracle signers to sign prices for a malicious feed ID like "BTC[null]MALICIOUS". The signed payload passes all the verification in `process_payload()`, but when converted to a string for storage at `get_prices_from_payload`, it becomes "BTC" overwriting the legitimate BTC price.

Recommendation Implement validation in `feed_to_string()` to detect and reject `FeedId` values containing null bytes in unexpected positions. The function should verify that the `FeedId` byte array consists only of non-zero bytes followed by trailing zeros, reverting if null bytes appear within the valid data portion.

Developers Response The code in `feed_to_string` now trim the byte array from left to right and right to left until the first non-zero byte is found which is consistent on how the `FeedId` type is built from a byte array, preventing that two different `FeedIds` collide to the same string.

5.1.3 V-RED-VUL-003: Two-step ownership transfer pattern not followed

Severity	Warning	**Commit**	726ac4
Type	Usability Issue	**Status**	Fixed
Location(s)	redstone-oracles-monorepo/packages/[...]/ownable.rs:10		
Confirmed Fix At	https://github.com/redstone-finance/redstone-oracles-monorepo/commit/e7a9bc257e364cff3b0e9c74ec3894c2ac95f4c8, e7a9bc2		

Description The ownership transfer process does not follow a two-step pattern; instead, ownership is transferred in a single step, immediately revoking the previous owner's rights.

Impact Mistakenly transferring ownership (e.g., due to a typo in the address) can hand control to an address that can't handle it; a two-step ownership transfer mitigates this risk by requiring the proposed new owner to actively accept via their own contract call.

Recommendation It is recommended to adopt a two-step ownership transfer process that requires the new owner to explicitly accept before finalization, preventing unintended transfers and ensuring the recipient is aware and capable of assuming the ownership responsibilities.

Developers Response The developers implemented the suggested recommendation.

5.1.4 V-RED-VUL-004: Missing ECDSA signature parameter validation

Severity	Warning	**Commit**	`ff1de9`
Type	Data Validation	**Status**	Fixed
Location(s)	`rust-sdk/crates/redstone/src/crypto/mod.rs:47-51`		
Confirmed Fix At	`https://github.com/redstone-finance/rust-sdk/pull/62`, `https://github.com/redstone-finance/rust-sdk/pull/63`, 3257c1a, be93165		

Description The RedStone protocol validates data packages by recovering signer addresses from ECDSA signatures. The signature verification process occurs in the `Crypto::recover_address()` method in `crypto/mod.rs`, which extracts a 65-byte signature and recovers the signer address through public key recovery.

The current implementation performs partial signature validation through `check_signature_malleability()`, which only verifies that the `s` parameter satisfies `s <= n/2`, where `n` is the order of the secp256k1 elliptic curve group. This check prevents high-s malleability but does not enforce the complete ECDSA parameter requirements specified in FIPS 186-5.

According to FIPS 186-5, valid ECDSA signatures must satisfy `1 <= r <= n - 1, 1 <= s <= n-1`. In this way, the implementation currently lacks explicit validation for:

1. `r != 0` - ensures the signature depends on the private key
2. `s != 0` - ensures the signature is non-trivial
3. `r < n` - prevents signature malleability through equivalent representations

Consider a scenario where a signature has `r = n`. After reduction modulo `n`, this becomes `r = 0 (mod n)`, making the signature independent of the private key. Without the `r < n` check, a zero check on the unreduced value would not catch this case. Similarly, if `r >= n`, an attacker could craft malleable signatures `(r, s)` and `(r+n, s)` that both verify successfully but have different binary representations.

The implementation delegates signature parsing and recovery to underlying libraries. While these libraries may perform some validation internally, relying on implicit library behavior creates compliance and security risks, particularly when integrating with different blockchain environments that may use alternative cryptographic implementations.

Impact The missing parameter validation creates three primary vulnerabilities:

1. **Non-compliance with cryptographic standards**: The implementation violates FIPS 186-5 requirements, which explicitly mandate rejection of signatures where `r` or `s` fall outside `[1, n-1]` in the first verification step.
2. **Signature malleability**: Attackers can craft alternative valid signatures `(r+n, s)` from legitimate signatures `(r, s)`, creating multiple valid signature representations for the same message. This breaks signature uniqueness assumptions and could enable replay attacks or circumvent signature-based deduplication mechanisms.
3. **Private-key-independent signatures**: Signatures with `r = 0` do not depend on the signer's private key, allowing forgery of signatures that pass verification without knowledge of the signing key.

Recommendation Implement explicit ECDSA parameter validation in `check_signature_malleability()` to enforce FIPS 186-5 compliance before signature recovery. Extract the `r` parameter from the signature and verify:

1. `r != 0`
2. `s != 0`
3. `r < n`

Developers Response The developers have yet to respond with acknowledgment or fixes.

5.1.5 V-RED-VUL-005: Informational: Code Quality and Best Practice Issues

Severity	Info	**Commit**	726ac4
Type	Maintainability	**Status**	Acknowledged
Location(s)	▶ `redstone-oracles-monorepo/packages/[...]/lib.rs:38, 38-39`		
Confirmed Fix At	N/A		

Description Multiple minor code quality and best practice issues have been identified in the codebase:

1. Typo in Error Message

In `rust-sdk/crates/redstone/src/types/mod.rs`, there is a typographical error in the panic message: `panic!("Number to big: {:?} digits", self.len())`. The error message should read "Number too big" instead of "Number to big".

1. Soroban Contract Initialization Pattern

It is identified that RedStoneAdapter and RedStonePriceFeed contracts are using the `init()` function to set the initial values for `owner` and `feed_ids`. However, Soroban contracts can be initialized directly with the __constructor() function since protocol version 22.

Impact Not using constructors can unintentionally expose contracts to front-running risks, where malicious actors can abuse their knowledge of pending transactions to set a different owner or feed id. Furthermore, using constructors also makes contracts more efficient by reducing their size.

Recommendation

1. Fix the typo: Change "Number to big" to "Number too big" in the error message.
2. Use Soroban constructors: It is recommended to make use of the __constructor() function to initialize contracts in `RedStoneAdapter` and `RedStonePriceFeed` implementations to prevent initialization front-running and improve contract efficiency.

Developer Response The developers acknowledged the finding but decided to not introduce code changes.

A Appendix

In addition to the confirmed findings, this report includes an appendix with issues that were ultimately classified as intended behavior or invalid after discussions with the development team. We chose to include these items in the report for two main reasons:

- **Transparency.** To document the scope of our review and ensure that all potential concerns identified during the audit are visible to stakeholders.
- **Context.** To provide the development team with a clear record of which findings were analyzed, discussed, and intentionally dismissed, avoiding future confusion if similar questions arise.

A.1 Intended Behavior

A.1.1 V-RED-APP-VUL-001: Missing price deviation checks enable attacks with less than 50% of the trusted signers

Severity	Low	**Commit**	`726ac4`
Type	Data Validation	**Status**	Intended Behavior
Location(s)	▶ `redstone-oracles-monorepo/packages/[...]/config.rs:32` ▶ `rust-sdk/crates/redstone/src/core/aggregator.rs:61-86`		

Description The RedStone adapter contract implements an oracle system that aggregates price data from multiple trusted signers. The system is configured with 5 authorized signers and requires a minimum of 3 valid signatures to accept a price update (`signer_count_threshold: 3`). When price updates are submitted via the `write_prices()` function, the payload is processed through `get_prices_from_payload()`, which invokes the RedStone SDK's `process_payload()` function. This SDK function verifies that at least 3 signatures from the authorized signer set are valid, then calculates the median of the signed price values to determine the final price stored on-chain.

The contract does not implement any validation to ensure price values from different signers are within reasonable proximity of each other. The median calculation tolerates less than 50% of input values being malicious. When only 3 signatures are provided (the minimum threshold), 2 malicious signers constitute 67% of the data points, enabling them to control the median output. Without deviation checks, these 2 signers can submit identical extreme values that override the honest signer's price.

For example, if the honest price is $100 and two malicious signers submit $1, the median of [$1, $1, $100] equals $1. The lack of validation against price deviation means this manipulation succeeds even though an honest signer participated. Notably, these 2 malicious signers represent only 40% of the total 5-signer set, yet they can fully compromise the oracle's integrity.

Impact Two compromised signers can manipulate any price feed to arbitrary values by submitting identical malicious prices. The absence of deviation checks means that even honest signer participation cannot prevent the attack when malicious signers form a simple majority of the minimum required signatures even if they do not represent a majority of the complete set of trusted signers.

Recommendation Implement price deviation validation in the RedStone SDK or adapter contract that verifies all submitted price values are within an acceptable percentage threshold of each other before accepting the aggregated result. This ensures that when an honest signer submits a legitimate price that significantly differs from colluding malicious signers, the transaction fails rather than accepting the manipulated median. With this control in place, attackers would need to compromise at least 3 signers to successfully manipulate prices (to outvote the deviation check triggered by honest signers), which represents a majority of the total 5-signer set and aligns with standard Byzantine fault tolerance assumptions.

Developers Response The developers acknowledged the reported behavior and confirmed that it is intentional and aligned with the protocol's design.

Having defined trusted_signers, it is assumed that the data is signed and not manipulated at any stage, from signing to sending to the contract. The contract is part of the entire system where data is verified at multiple levels using a similar mechanism as proposed, off-chain. The operating principle of the entire system across other blockchains, including the entire EVM ecosystem, is calculating the median on-chain from trusted_signers data as an additional rather than primary security measure. While we understand this threat, the probability of data manipulation in the system is practically zero in practice. We will discuss this further; however, at this moment this approach is consistent with the operation on other blockchains.

A.1.2 V-RED-APP-VUL-002: Price Feeds lack emergency pause mechanism for compromised adapters

Severity	Warning	**Commit**	726ac4
Type	Data Validation	**Status**	Intended Behavior
Location(s)	redstone-oracles-monorepo/packages/[...]/lib.rs:97-104		

Description The RedStone oracle system on Stellar consists of two primary contracts: the `RedStoneAdapter` contract that receives and stores price data from RedStone oracles, and multiple `RedStonePriceFeed` contracts that act as individual price feed proxies for specific assets. Each `RedStonePriceFeed` contract is initialized with a specific `feed_id`(such as "BTC" or "ETH") and queries the shared `RedStoneAdapter` contract to retrieve price data via the `read_price_data_for_feed()` function.

The `RedStonePriceFeed` contract implements several read functions (`read_price()`, `read_timestamp()`, `read_price_data()`) that all ultimately call `read_price_data()`. This function creates a client for the hardcoded adapter address and invokes `read_price_data_for_feed()` on the `RedStoneAdapter` contract. The adapter address is immutably defined as a constant string and cannot be modified after deployment, unless the contract is upgraded to a new WASM binary .

The `RedStoneAdapter` contract has no mechanism to signal to `RedStonePriceFeed`contracts that it has been compromised or to revert price queries during emergency situations. Once price data is stored in the adapter's persistent storage, it will be served to all requesting feed contracts regardless of whether the data is trustworthy.

Impact In the event of a vulnerability in the Rust SDK, or any other scenario where prices are not reliable, there is no mechanism to prevent downstream consumers from continuing to use potentially compromised or inaccurate/incorrect price data.

Recommendation Implement a circuit breaker mechanism in the `RedStoneAdapter` contract that includes a boolean status flag indicating whether the adapter is in a healthy or paused state. Add an owner-controlled function to toggle this status flag. Modify the `read_price_data_for_feed()` and related read functions to check this status flag before returning price data.

Additionally, consider implementing this status check in the `RedStonePriceFeed` contracts' `read_price_data()` function to query the adapter's status before consuming prices. This allows the protocol team to immediately halt all price consumption across the ecosystem in response to security incidents while investigating and deploying fixes.

Developers Response The developers acknowledged the reported behavior and confirmed that it is intentional and aligned with the protocol's design.

We don't have a flag to mark an adapter as compromised for price feeds. However, even though price feed addresses remain unchanged, we can update the adapter address by upgrading the price feed (for example, by pointing it to a newly deployed adapter). This ensures that storage from the compromised adapter is no longer visible to clients.

A.1.3 V-RED-APP-VUL-003: Signer addresses cannot be rotated without contract upgrade

Severity	Warning	**Commit**	`726ac4`
Type	Maintainability	**Status**	Intended Behavior
Location(s)	`redstone-oracles-monorepo/packages/[...]/config.rs:40-46`		

Description The RedStone adapter contract verifies price feed data signatures using a set of trusted signer addresses defined in the configuration. These signer addresses are stored in the constant `REDSTONE_PRIMARY_PROD_ALLOWED_SIGNERS`, which contains five hardcoded 20-byte Ethereum-style addresses.

When the payload is processed through `get_prices_from_payload()`, the function calls `STELLAR_CONFIG.redstone_config()`, which converts the hardcoded signers into `SignerAddress` objects via `redstone_signers()`. These signers are then used to verify the signatures on price feed payloads. Because these signer addresses are defined as compile-time constants, they are embedded directly into the deployed WASM binary.

This design prevents both emergency revocation of compromised signers and routine key rotation. If a signer's private key is compromised or simply reaches its recommended rotation period, there is no runtime mechanism to revoke that signer or add a replacement. The only remediation path is to upgrade the contract with a new WASM binary via the `upgrade()` function. Contract upgrades involve deploying new bytecode, which introduces operational delays and risks of their own.

Impact Signer keys cannot be revoked or rotated, allowing attackers to potentially inject fraudulent price data into the oracle system if they control enough signers. As discussed in a separate finding compromising fewer than 50% of the total signers is sufficient to manipulate the system. This significantly lowers the security threshold, making the inability to rapidly rotate compromised keys even more important. The inability to rapidly rotate signers also means the protocol must rely on slow contract upgrade processes during security emergencies, extending the window of vulnerability.

Recommendation Implement a runtime signer management system that stores authorized signers in contract storage rather than as compile-time constants. This enables both routine key rotation and emergency revocation without full contract redeployment. However, signer management functions introduce their own security considerations around authorization and timing controls.

Consider implementing asymmetric protections based on the security implications of each operation. For adding new signers via `add_signer()`, apply a timelock mechanism to prevent rapid unauthorized changes, as malicious signer additions could compromise oracle integrity. For removing signers via `remove_signer()`, allow immediate execution to enable rapid emergency response when keys are compromised, while ensuring the removal logic validates that the `signer_count_threshold` remains achievable after removal to prevent accidental denial of service.

Developers Response The developers acknowledged the reported behavior and confirmed that it is intentional and aligned with the protocol's design.

That's intended. We don't want to have it changeable by an admin function, safety reasons. So changing the config or signers will be performed by contract upgrades.

A.1.4 V-RED-APP-VUL-004: DataPackage signatures lack domain separator for chain identification

Severity	Warning	**Commit**	`ff1de9`
Type	Data Validation	**Status**	Intended Behavior
Location(s)	`rust-sdk/crates/redstone/src/[...]/payload_decoder.rs:89`		

Description The RedStone Rust SDK provides a multi-chain oracle solution supporting chains such as EVM chains, Solana and Soroban. The `PayloadDecoder` in `payload_decoder.rs` processes oracle payloads by decoding `DataPackage` structures and verifying their signatures. During `trim_data_package()`, the SDK extracts `signable_bytes` consisting solely of the data package contents (data points, timestamp, value size, and point count) without any chain-specific domain separator. The `Crypto::recover_address()` function then computes `keccak256(message)` directly on these bytes to recover the signer's address. This means the signature is computed over identical data regardless of which blockchain the payload is used on.

For example, a `DataPackage` signed for an EVM chain contains bytes representing: `[data_points][timestamp][value_size][point_count]`. When this same payload is submitted to Soroban, the `signable_bytes` extracted in `trim_data_package()` will be identical, causing `recover_address()` to successfully recover the same signer address, passing the validation on both chains.

This design allows the same signed oracle data to be accepted as valid on multiple blockchains whenever they share the same authorized signer set. While this may be intentional, it creates a dependency: all blockchain implementations must process payloads identically, or the same valid signed `DataPackages` could produce different oracle values across chains.

Impact Valid oracle payloads can be replayed across different blockchains that share the same authorized signer configuration. If payload processing logic differs between chain implementations (even subtly), the same signed data could produce different oracle values on different chains, creating cross-chain oracle inconsistencies.

Recommendation Add a chain identifier or domain separator to the signed message to make signatures chain-specific. Alternatively, if cross-chain payload sharing is intentional, document this architectural decision in all implementation guides and enforce semantic equivalence testing across all supported blockchain platforms.

Developers Response The developers acknowledged the reported behavior and confirmed that it is intentional and aligned with the protocol's design.

This is our core design choice. We want any single payload to be valid on every chain we support. We have some common testing between different chains which we are extending all the time - and documentation about our architecture.

A.2 Invalid Issues

A.2.1 V-RED-APP-VUL-005: Overflow check not enabled for release builds

Severity	Warning	**Commit**	`ff1de9`
Type	Maintainability	**Status**	Invalid
Location(s)	`rust-sdk/crates/redstone/Cargo.toml`		

Description By default, Rust enables integer overflow checks in debug builds but disables them in release builds for performance, unless `overflow-checks = true` is present `Cargo.toml` . It is identified `overflow-checks = true` is missing in `Cargo.toml` (located in `redstone/Cargo.toml`)

Impact Arithmetic operations can overflow (or underflow) and wrap silently instead of aborting, leading to unexpected behavior and results.

Proof of Concept Run the following test from within the `/crates/redstone` folder. (For the purpose of verifying the bug, the test was included in `/crates/redstone/src/core/config.rs` file)

```
    #[test]
    fn test_overflow_test() {
        let mut x: u32 = u32::MAX;
        x += 1;

        assert_eq!(x, 0);
    }
```

Run the test with the following:

```
cargo test --release --package redstone test_overflow_test
```

Recommendation It is recommended to include `overflow-checks = true` in the `Cargo.toml` file.

Why Invalid It turns out that Cargo only considers the profile settings defined in the `Cargo.toml` manifest at the root of the workspace. Profile settings defined in dependencies are ignored. Since the Rust SDK is a dependency of the Soroban contracts workspace, which sets `overflow-checks = true`, the final binary will be built with overflow checks enabled. Developers building the package standalone can manually add the check if needed.

Reference: https://doc.rust-lang.org/cargo/reference/profiles.html#profiles