# Soroban-SDK

*Stellar*

HALBORN

# Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ABOUT THIS REPORT

ASSESSMENT TYPE
Assurance Assessment

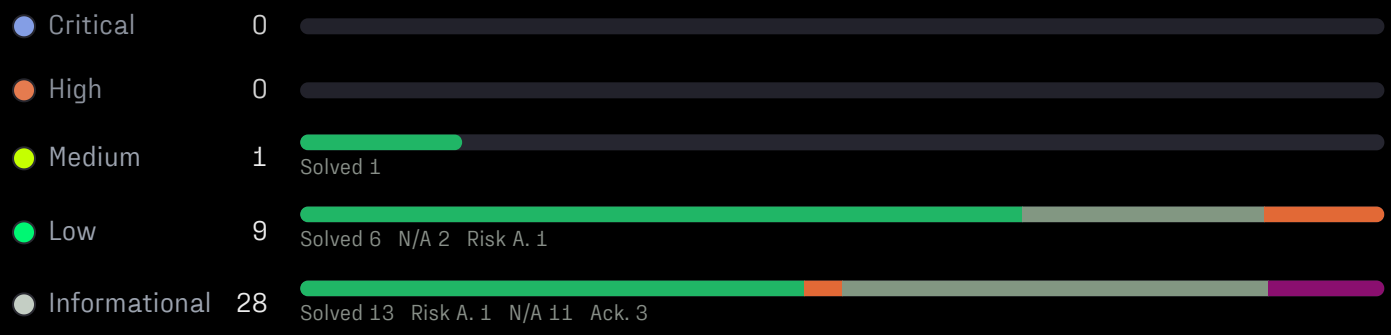
DATE OF ENGAGEMENT
Feb 9, 2026 - Mar 7, 2026

SOURCE CODE
rs-soroban-sdk

ASSESSED COMMIT ID
86c50a1

SEVERITY BREAKDOWN

| 38 | 20 | 2 | 3 | 13 |
|---|---|---|---|---|
| All Findings | Solved | Risk Accepted | Acknowledged | Not Applicable |




## INTRODUCTION

The project team engaged Halborn to conduct a security assessment of Soroban SDK v25.1.0, the Rust SDK used to build smart contracts for the Soroban runtime on Stellar. The assessment covered the core soroban-sdk crate, proc-macro and code-generation surfaces, token and asset helper libraries, spec and metadata extraction utilities, ledger snapshot tooling, and migration guidance intended for downstream developers. Because this project is an SDK rather than a single deployed application, the security boundary extends beyond runtime correctness and includes generated code, documentation patterns, test utilities, and off-chain tooling that processes untrusted contract artifacts.

At a high level, the Soroban SDK ecosystem supports four primary workflows: contract authoring in Rust, macro expansion and Wasm generation, execution inside the Soroban host/runtime, and downstream tooling that extracts specs, generates clients, and simulates ledger state for testing and integration. As a result, correctness must hold across the host/guest boundary, generated code must faithfully represent contract interfaces, and official examples must not encourage unsafe patterns that downstream developers may copy into production contracts.

Overall, the assessed version demonstrated a solid baseline security posture. The review did not confirm critical or high-severity issues in the core SDK, and the findings were concentrated

primarily in low-severity and informational hardening opportunities rather than systemic design failure. The most significant issue involved an official Pausable migration example whose default trait methods lack authorization, creating a copy-pasteable insecure pattern with ecosystem-wide downstream risk. Beyond that, the report primarily identified trap-prone APIs, sharp edges at contract call boundaries, brittle proc-macro and code-generation behavior, and off-chain tooling paths that can be forced into excessive resource consumption when handling malformed or adversarial inputs.

## Key themes observed across findings

- Insecure or incomplete official examples and migration guidance, where downstream users may reasonably assume a pattern is safe to adopt as-is.
- Host/guest boundary sharp edges, especially around panic-oriented conversions, cross-contract call semantics, storage TTL handling, and metadata access patterns that can unexpectedly trap.
- Schema and compatibility issues in token, SAC, and event-related abstractions, where fragmented or evolving formats can create downstream integration and indexing risk.
- Code-generation and proc-macro robustness gaps, including malformed identifier handling, UTF-8 truncation, brittle type mapping, and compile-time panics instead of actionable diagnostics.
- Off-chain tooling DoS surfaces, especially in Wasm spec/meta extraction, snapshot parsing, and artifact-processing paths that operate on untrusted inputs before hard bounds are enforced.
- Crypto and type-safety pitfalls, including non-canonical field/scalar encodings, hazmat-style APIs, and guarantees that are weaker in practice at the contract boundary than they appear at the API level.
- Test and production divergence, where mock auth helpers, registration utilities, and local environment behavior can hide real-world issues or create false confidence in integration behavior.

## Main recommendations

- Remove or harden any official examples that can be copied into vulnerable production code, and add prominent warnings wherever the SDK intentionally leaves critical security logic to the contract author.
- Replace panic-oriented behaviors in proc-macros, code generators, and developer tooling with structured, actionable errors wherever feasible.

- Enforce strict size and complexity bounds before parsing, copying, or decoding untrusted Wasm, metadata, base64, or snapshot inputs.
- Tighten macro and code-generation invariants around identifiers, type resolution, event schemas, and naming collisions so generated artifacts are both correct and predictable.
- Clarify and, where possible, redesign trap-prone runtime APIs so developers can choose recoverable error handling paths rather than inheriting brittle defaults.
- Reduce misleading differences between test-only helpers and production semantics, or document those differences so clearly that misuse is unlikely.
- Standardize event and spec behaviors across token-related surfaces to reduce ecosystem fragmentation for clients, indexers, and downstream contracts.

## System Architecture

The Soroban SDK is best understood as a layered smart contract development stack rather than a single deployed service. Developers write contracts in Rust using the SDK's core APIs and attribute macros. Those macros generate entrypoints, wrappers, clients, specs, and event machinery that are then compiled into Wasm for execution inside the Soroban host/runtime on Stellar. Around that core path, the project also includes off-chain libraries that read Wasm custom sections, generate Rust bindings, expose token and asset abstractions, and serialize or restore ledger state for testing and integration workflows.

## Main components

- Contract authors and downstream applications
  - Use the SDK's Rust APIs, macros, generated clients, and helper types to implement contract logic, events, authorization flows, and storage access.
- Proc-macro and code-generation layer
  - The soroban-sdk-macros crate provides annotations such as #[contract], #[contractimpl], #[contractclient], #[contracttype], and #[contractevent], which transform Rust source into Soroban-compatible entrypoints, wrapper code, specs, and generated client interfaces.
- Core runtime-facing SDK
  - The soroban-sdk crate provides the contract-facing surface for Env, auth, storage, deployment, events, collections, numeric and bytes types, logging, crypto wrappers, PRNG

functionality, and conversion utilities that operate across the host/guest boundary.

- Token and asset helper layer
  - Libraries such as soroban-token-sdk, soroban-token-spec, and stellar-asset-spec provide reusable token-focused abstractions, event definitions, and metadata/spec support for token contracts and adjacent integrations.

- Spec and metadata extraction layer
  - soroban-spec and soroban-meta parse Wasm custom sections to recover contract specs and metadata, while soroban-spec-rust turns those artifacts into Rust traits, clients, and related generated bindings for downstream consumers.

- Test and snapshot infrastructure
  - testutils and soroban-ledger-snapshot support local contract testing, state restoration, fixture handling, and developer simulation of ledger-backed behavior outside production deployment.

- Soroban host/runtime on Stellar
  - Compiled Wasm contracts ultimately execute against host-provided storage, auth, budget, deployment, and value-conversion semantics. This host boundary is one of the most important security interfaces in the entire SDK.

- External tooling and CI consumers
  - Build systems, code generators, indexers, internal developer tooling, and CI pipelines may process Wasm artifacts, specs, metadata, generated clients, and snapshots, making off-chain robustness and bounded parsing an important part of the security model.

## Primary workflows

1. Contract authoring and compilation
   - A developer writes Rust contract code using SDK types and macros. Macro expansion generates contract entrypoints and metadata-related helpers, and the resulting crate is

compiled to Wasm.

2. Runtime execution

- The compiled Wasm contract runs inside the Soroban host, which provides storage, authorization, event publication, deployment support, cryptographic helpers, and value conversion primitives.

3. Spec extraction and client generation

- Off-chain tooling reads the Wasm's custom sections to extract specs and metadata, then uses those artifacts to generate Rust bindings, client wrappers, or other integration-facing outputs.

4. Testing and state simulation

- Local environments and ledger snapshots are used to simulate contract execution, replay state, inspect behavior, and validate logic without a live deployment.

## Trust boundaries

- Developer source and documentation into macro/code-generation paths
    - If examples, migration snippets, or generated wrappers encode unsafe assumptions, those assumptions can propagate into many downstream contracts.

- Contract code into host-provided runtime semantics
    - The SDK surface often presents high-level Rust ergonomics, but safety ultimately depends on how those APIs map onto host behavior, conversion rules, and trap conditions.

- Untrusted Wasm, specs, metadata, and snapshots into off-chain tooling
    - Any system that parses third-party artifacts must treat them as attacker-controlled and enforce hard bounds before allocating or decoding.

- Test-only helpers versus production semantics

- Mock auth, registration helpers, local environment behavior, and snapshot defaults can diverge from production and obscure real integration risk if used incautiously.

## Key architectural asymmetry

The project has a meaningful split between runtime-facing SDK components and off-chain tooling/components. Runtime issues can directly affect deployed contracts and downstream application semantics, while tooling issues more often affect CI, code generation, artifact processing, and developer reliability. However, because this is a foundational SDK, even documentation flaws or tooling edge cases can scale into ecosystem-wide risk if they are copied broadly or embedded into automation pipelines.

## Risk Methodology

Halborn assesses the severity of findings using either the Common Vulnerability Scoring System (CVSS) framework or the Impact/Likelihood Risk scale, depending on the engagement. CVSS is an industry standard framework for communicating characteristics and severity of vulnerabilities in software. Details can be found in the CVSS Specification Document published by F.I.R.S.T.

Vulnerabilities or issues observed by Halborn scored on the Impact/Likelihood Risk scale are measured by the LIKELIHOOD of a security incident and the IMPACT should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

### RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

### RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

| CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL |
|---|---|---|---|---|

**10** - CRITICAL
**9 - 8** - HIGH
**7 - 6** - MEDIUM
**5 - 4** - LOW
**3 - 1** - VERY LOW AND INFORMATIONAL

## Scope

REPOSITORY

(a) Repository: rs-soroban-sdk
(b) Assessed Commit ID: 86c50a1ea4f87b40add3064ff9df95c7553565c5
(c) Items in scope:

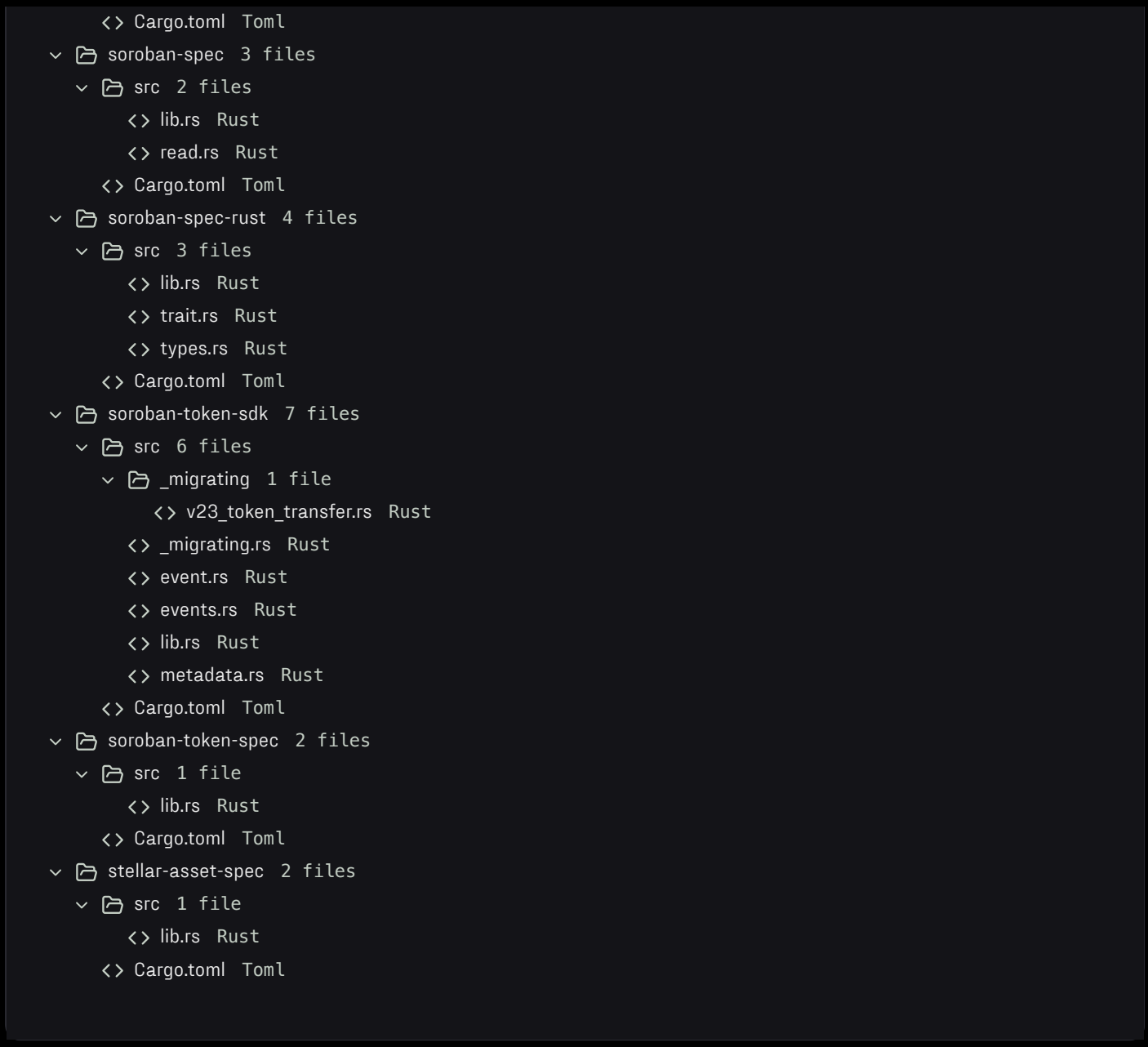
- soroban-ledger-snapshot 2 files
  - src 1 file
    - lib.rs Rust
  - Cargo.toml Toml
- soroban-meta 3 files
  - src 2 files
    - lib.rs Rust
    - read.rs Rust
  - Cargo.toml Toml
- soroban-sdk 42 files
  - src 40 files
    - _migrating 7 files
      - v23_archived_testing.rs Rust
      - v23_contractevent.rs Rust
      - v25_bn254.rs Rust
      - v25_contracttrait.rs Rust
      - v25_event_testing.rs Rust
      - v25_poseidon.rs Rust
      - v25_resource_limits.rs Rust
    - crypto 3 files
      - bls12_381.rs Rust
      - bn254.rs Rust
      - utils.rs Rust
    - _migrating.rs Rust
    - address_payload.rs Rust
    - address.rs Rust
    - alloc.rs Rust

- arbitrary_extra.rs Rust
- auth.rs Rust
- bytes.rs Rust
- constructor_args.rs Rust
- crypto.rs Rust
- deploy.rs Rust
- env.rs Rust
- error.rs Rust
- events.rs Rust
- iter.rs Rust
- ledger.rs Rust
- lib.rs Rust
- logs.rs Rust
- map.rs Rust
- muxed_address.rs Rust
- num.rs Rust
- prng.rs Rust
- storage.rs Rust
- string.rs Rust
- symbol.rs Rust
- token.rs Rust
- try_from_val_for_contract_fn.rs Rust
- tuple.rs Rust
- unwrap.rs Rust
- vec.rs Rust
- xdr.rs Rust
- build.rs Rust
- Cargo.toml Toml

soroban-sdk-macros 22 files

src 21 files

- arbitrary.rs Rust
- attribute.rs Rust
- derive_args.rs Rust
- derive_client.rs Rust
- derive_contractimpl_trait_default_fns_not_overridden.rs Rust
- derive_contractimpl_trait_macro.rs Rust
- derive_enum_int.rs Rust
- derive_enum.rs Rust
- derive_error_enum_int.rs Rust
- derive_event.rs Rust
- derive_fn.rs Rust
- derive_spec_fn.rs Rust
- derive_struct_tuple.rs Rust
- derive_struct.rs Rust
- derive_trait.rs Rust
- doc.rs Rust
- lib.rs Rust
- map_type.rs Rust
- path.rs Rust
- symbol.rs Rust
- syn_ext.rs Rust




REMEDIATION COMMIT ID:

https://github.com/stellar/rs-soroban-sdk/pull/1729
https://github.com/stellar/rs-soroban-sdk/pull/1764
https://github.com/stellar/rs-soroban-sdk/pull/1788
https://github.com/stellar/rs-soroban-sdk/pull/1780
https://github.com/stellar/rs-soroban-sdk/pull/1771
https://github.com/stellar/rs-soroban-sdk/pull/1794
https://github.com/stellar/rs-soroban-sdk/pull/1796
https://github.com/stellar/rs-soroban-sdk/pull/1767
https://github.com/stellar/rs-soroban-sdk/pull/1810
https://github.com/stellar/rs-soroban-sdk/pull/1798
https://github.com/stellar/rs-soroban-sdk/pull/1774
https://github.com/stellar/rs-soroban-sdk/security/advisories/GHSA-x2hw-px52-wp4m

https://github.com/stellar/rs-soroban-sdk/pull/1733
https://github.com/stellar/rs-soroban-sdk/pull/1792
https://github.com/stellar/rs-soroban-sdk/pull/1803
https://github.com/stellar/rs-soroban-sdk/pull/1804
https://github.com/stellar/rs-soroban-sdk/pull/1809
https://github.com/stellar/rs-soroban-sdk/pull/1806
https://github.com/stellar/rs-soroban-sdk/pull/1761

Out-of-Scope: New features/implementations after the remediation commit IDs.

## Assessment Summary & Findings Overview

| # | TITLE | SEVERITY | SCORE | STATUS |
|---|---|---|---|---|
| HAL-001 | contractimpl trait dispatch can call inherent methods instead of trait methods on name collisions | Medium | 5.9 | SOLVED 02/13/2026 |
| HAL-002 | Official Pausable contracttrait example is unauthenticated (promotes a vulnerable pattern) | Low | 3.3 | SOLVED 03/16/2026 |
| HAL-003 | Proc-macro spec emission uses unwrap() on XDR serialization (compiler panic instead of compile error) | Low | 2.9 | NOT APPLICABLE |
| HAL-004 | Macro type mapping matches built-ins by last path segment (shadow-type ABI/spec corruption) | Low | 2.9 | SOLVED 03/31/2026 |
| HAL-005 | Associated type flattening is one-pass and non-recursive (Self::TypeAlias chains) | Low | 2.9 | SOLVED 03/19/2026 |
| HAL-006 | Token/SAC event schema fragmentation and legacy publisher exposure (mis-indexing risk) | Low | 2.9 | RISK ACCEPTED 03/24/2026 |
| HAL-007 | build.rs Rust-version guard is ineffective under cross-compilation | Low | 2.9 | SOLVED 03/18/2026 |
| HAL-008 | Macro parsing discards outer attributes on trait/impl inputs (#[cfg] not propagated) | Low | 2.9 | NOT APPLICABLE |

| # | TITLE | SEVERITY | SCORE | STATUS |
|---|---|---|---|---|
| HAL-009 | contractevent(data_format="single-value") does not enforce single-field and can generate invalid Rust | Low | 2.9 | SOLVED 03/26/2026 |
| HAL-010 | LedgerSnapshot writes are non-atomic and can truncate outputs on failure | Low | 2.9 | SOLVED 03/26/2026 |
| HAL-011 | FromXdr guest or WASM error handling can trap before returning Err | Informational | — | SOLVED 03/16/2026 |
| HAL-012 | LedgerSnapshot uses linear scans for lookup and update (large-snapshot performance footgun) | Informational | — | RISK ACCEPTED 04/30/2026 |
| HAL-013 | soroban-spec-rust generator assumes trusted spec identifiers when generating bindings | Informational | — | SOLVED 04/24/2026 |
| HAL-014 | Cross-contract call helpers favor documented panics and coarse error reporting over recoverability | Informational | — | NOT APPLICABLE |
| HAL-015 | testutils auth helpers can mask authorization bugs (mock_all_auths, authorize_as_current_contract) | Informational | — | SOLVED 03/27/2026 |
| HAL-016 | Typed collection convenience iterators unwrap conversion failures in raw-Val and storage-driven flows | Informational | — | SOLVED 03/19/2026 |
| HAL-017 | Crypto scalar wrapper construction does not enforce canonical encoding invariants | Informational | — | SOLVED 03/06/2026 |
| HAL-018 | Token metadata helpers assume initialized state and do not validate metadata values | Informational | — | NOT APPLICABLE |
| HAL-019 | contractimport! does not require SHA-256 pinning and accepts arbitrary paths (integrity/reproducibility footgun) | Informational | — | NOT APPLICABLE |
| HAL-020 | BytesN convenience APIs have edge-case semantics and performance footguns | Informational | — | SOLVED 02/13/2026 |
| HAL-021 | Macro-generated identifiers and helper names can collide across generated surfaces | Informational | — | NOT APPLICABLE |

| # | TITLE | SEVERITY | SCORE | STATUS |
|---|---|---|---|---|
| HAL-022 | testutils function registry uses global mutable state and panics on poisoned mutex | Informational | — | NOT APPLICABLE |
| HAL-023 | Debug/test-only helpers can panic unexpectedly (Logs::add, native String formatting) | Informational | — | NOT APPLICABLE |
| HAL-024 | Panic-oriented parsing helpers can trap on invalid caller-supplied strings (Address::from_str, MuxedAddress::from_str, Symbol::new) | Informational | — | NOT APPLICABLE |
| HAL-025 | Storage TTL helpers assume host invariants and trap on invalid parameter flows | Informational | — | SOLVED 03/25/2026 |
| HAL-026 | register_* test utilities are not panic-safe and can leave Env state inconsistent | Informational | — | SOLVED 03/31/2026 |
| HAL-027 | WASM bump allocator is intentionally non-freeing (memory-growth footgun inside an invocation) | Informational | — | ACKNOWLEDGED 03/31/2026 |
| HAL-028 | secp256k1_recover parameter typo (recorvery_id) reduces clarity | Informational | — | SOLVED 03/31/2026 |
| HAL-029 | AddressPayload constructs XDR manually (hazmat/test feature; fragile encoding assumptions) | Informational | — | NOT APPLICABLE |
| HAL-030 | Generated WASM export names use a flat method-name namespace | Informational | — | SOLVED 03/30/2026 |
| HAL-031 | Token migration doc link is incorrect (Address links to MuxedAddress) | Informational | — | SOLVED 03/31/2026 |
| HAL-032 | PRNG is explicitly non-secret and validator-influenceable (misuse breaks security assumptions) | Informational | — | NOT APPLICABLE |
| HAL-033 | assert_in_contract! is test-only and is a no-op in production builds | Informational | — | SOLVED 03/31/2026 |
| HAL-034 | Token spec generation hardcodes XDR_LEN and panics at const-eval (brittle maintenance) | Informational | — | ACKNOWLEDGED 03/31/2026 |

| # | TITLE | SEVERITY | SCORE | STATUS |
|---|---|---|---|---|
| HAL-035 | soroban-meta parser scales XDR byte limits with input size | Informational | — | NOT APPLICABLE |
| HAL-036 | LedgerSnapshot::default embeds the current protocol version rather than an obviously empty sentinel | Informational | — | ACKNOWLEDGED 03/26/2026 |
| HAL-037 | LedgerSnapshot::update ignores ledger-info read results in testutils flows | Informational | — | NOT APPLICABLE |
| HAL-038 | Generated testutils try_ client functions ignore allow_non_root_auth setting | Informational | — | SOLVED 03/13/2026 |

## Findings & Tech Details

HAL-001 SOLVED

# contractimpl trait dispatch can call inherent methods instead of trait methods on name collisions

Medium · 5.9 AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:H/A:N

DESCRIPTION

When `#[contractimpl]` generates the exported entrypoint wrappers for a trait implementation, it resolves the target call as `<Type>::method` instead of `<Type as Trait>::method`.
In Rust, associated function lookup prefers inherent implementations over trait implementations when both exist with the same name. As a result, if a contract defines an inherent function and a trait implementation function with identical names, the generated Wasm entrypoint can silently dispatch to the inherent function rather than the trait function that the ABI/spec implies.
This can bypass logic that exists only in the trait implementation, including authorization or invariant checks. The issue also affects default trait functions routed through the same code-generation path.

### Code Location

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk-macros/src/derive_fn.rs:49-53
// Collect errors as they are encountered and emit them at the end.
let mut errors = Vec::<Error>::new();

let call = quote! { <#impl_ty>::#ident };

// file: rs-soroban-sdk-25.1.0/soroban-sdk-
macros/src/derive_contractimpl_trait_default_fns_not_overridden.rs:65-72
let mut output = quote! {};
output.extend(derive_pub_fns(
    &args.crate_path,
    &impl_ty,
    &fns,
    Some(trait_ident),
    &args.client_name,
));
```

### Impact

- Public contract entrypoints generated from trait implementations can invoke the wrong function body when an inherent method with the same name exists.
- If the trait implementation contains authorization or safety checks that the inherent method omits, external callers can reach behavior that contradicts the intended contract interface.
- Contracts compiled with vulnerable macro versions must be recompiled after upgrading, because the issue is in generated dispatch glue.

RECOMMENDATION

To mitigate this issue following improvements are recommended

- Upgrade to `soroban-sdk-macros` `25.1.1` or later and recompile affected contracts so generated wrappers use trait-qualified dispatch.
- Until upgraded, avoid defining inherent functions whose names collide with functions in `#[contractimpl]` trait implementations.
- Add a regression test covering a contract type that defines both an inherent method and a trait method with the same name.

REFERENCES

GHSA-4chv-4c6w-w254

REMEDIATION COMMENT

Addressed in: https://github.com/stellar/rs-soroban-sdk/pull/1729

**SOLVED:** Client remediated this issue by updating macro-generated trait dispatch to use fully qualified trait calls and by adding a regression test.

HAL-002 SOLVED

# Official Pausable contracttrait example is unauthenticated (promotes a vulnerable pattern)

Low · 3.3 AV:L/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:N

DESCRIPTION

The v25 migration guide includes a `Pausable` trait whose default `pause()` and `unpause()` implementations modify instance storage without any access control (e.g., `require_auth`). Because `#[contractimpl(contracttrait)]` exports trait default implementations as callable contract functions, copy-pasting this example verbatim will expose `pause()` / `unpause()` as publicly callable entrypoints.

## Code Location

[data-start="54"]

RUST

```
// file: rs-soroban-sdk-25.1.0/soroban-sdk/src/_migrating/v25_contracttrait.rs:54-66
//! #[contracttrait]
//! pub trait Pausable {
//!     fn is_paused(env: &Env) -> bool {
//!         env.storage().instance().has(&"paused")
//!     }
//!
//!     fn pause(env: &Env) {
//!         env.storage().instance().set(&"paused", &true);
//!     }
//!
//!     fn unpause(env: &Env) {
//!         env.storage().instance().remove(&"paused");
//!     }
```

## Impact

- Any external caller can pause/unpause the contract (authorization bypass + DoS).
- Ecosystem-wide risk: multiple contracts may replicate this insecure pattern from official migration docs.

RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Update the example to include explicit access control (e.g., load an `admin: Address` from instance storage and call `admin.require_auth()`).
- If the SDK intentionally cannot prescribe auth, add a prominent warning above the defaults stating they must be overridden with authorization checks.

REMEDIATION COMMENT

Addressed in: https://github.com/stellar/rs-soroban-sdk/pull/1764

**SOLVED:** Client remediated this issue by adding admin authorization to the pause and unpause defaults, initializing admin state in the constructor, and covering the example with tests.

HAL-003 NOT APPLICABLE

# Proc-macro spec emission uses unwrap() on XDR serialization (compiler panic instead of compile error)

Low · 2.9 AV:L/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:L

DESCRIPTION

Multiple macro derivations serialize spec entries to XDR using `to_xdr(...).unwrap()`. If serialization fails (limits exceeded, invalid intermediate state), the proc-macro panics, resulting in a hard compilation crash rather than an actionable diagnostic.

## Code Location

[data-start="77"]

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk-macros/src/derive_struct.rs:77-92
    // Generated code spec.
    let spec_gen = if spec {
        let spec_entry = ScSpecEntry::UdtStructV0(ScSpecUdtStructV0 {
            doc: docs_from_attrs(attrs),
            lib: lib.as_deref().unwrap_or_default().try_into().unwrap(),
            name: ident.to_string().try_into().unwrap(),
            fields: spec_fields.try_into().unwrap(),
        });
        let spec_xdr = spec_entry.to_xdr(DEFAULT_XDR_RW_LIMITS).unwrap();
        let spec_xdr_lit = proc_macro2::Literal::byte_string(spec_xdr.as_slice());
        let spec_xdr_len = spec_xdr.len();
        let spec_ident = format_ident!("__SPEC_XDR_TYPE_{}",
ident.to_string().to_uppercase());
        Some(quote! {
            #[cfg_attr(target_family = "wasm", link_section = "contractspecv0")]
            pub static #spec_ident: [u8; #spec_xdr_len] = #ident::spec_xdr();
```

## Impact

- Build-time DoS and poor diagnostics for edge-case contracts (large specs, large enums/structs/events).

RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Convert XDR serialization errors into `syn::Error` and emit `compile_error!`-based diagnostics rather than panicking.
- Add pre-checks for “too many fields/variants/params” with clear, user-facing error messages.

REMEDIATION COMMENT

**NOT APPLICABLE:** Client clarified that this is compiler diagnostic and failure-ergonomics feedback rather than a security finding, because invalid XDR values are not a valid contract build path.

HAL-004 SOLVED

# Macro type mapping matches built-ins by last path segment (shadow-type ABI/spec corruption)

Low · 2.9 AV:L/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:L

DESCRIPTION

The macro type-mapping logic recognizes SDK built-in types by matching only the last path segment (e.g., `Address`, `Bytes`, `Symbol`).
A user-defined type such as `my_mod::Address` may therefore be silently treated as the SDK built-in type, producing corrupted/misleading ABI specs and potentially causing runtime conversion failures at call boundaries.

## Code Location

[data-start="34"]

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk-macros/src/map_type.rs:34-58
        Type::Path(TypePath {
            qself: None,
            path: Path { segments, .. },
        }) => {
            match segments.last() {
                Some(PathSegment {
                    ident,
                    arguments: PathArguments::None,
                }) => match &ident.to_string()[..] {
                    "Val" => Ok(ScSpecTypeDef::Val),
                    "u64" => Ok(ScSpecTypeDef::U64),
                    "i64" => Ok(ScSpecTypeDef::I64),
                    "u32" => Ok(ScSpecTypeDef::U32),
                    "i32" => Ok(ScSpecTypeDef::I32),
                    "u128" => Ok(ScSpecTypeDef::U128),
                    "i128" => Ok(ScSpecTypeDef::I128),
                    "U256" => Ok(ScSpecTypeDef::U256),
                    "I256" => Ok(ScSpecTypeDef::I256),
                    "bool" => Ok(ScSpecTypeDef::Bool),
                    "Symbol" => Ok(ScSpecTypeDef::Symbol),
                    "String" => Ok(ScSpecTypeDef::String),
                    "Error" => Ok(ScSpecTypeDef::Error),
                    "Bytes" => Ok(ScSpecTypeDef::Bytes),
                    "Address" => Ok(ScSpecTypeDef::Address),
                    "MuxedAddress" => Ok(ScSpecTypeDef::MuxedAddress),
```

## Impact

- ABI/spec mismatch that can mislead clients/tooling and cause call-boundary traps or interoperability failures.

RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Match built-in types by full, known path (e.g., `soroban_sdk::Address`), or treat any multi-segment path as user-defined.
- Emit a clear compile error when non-SDK paths use reserved built-in short names.

REMEDIATION COMMENT

Addressed in: https://github.com/stellar/rs-soroban-sdk/pull/1788

**SOLVED:** Client successfully remediated this issue by implementing compile-time validation that rejects contract types whose names collide with reserved SDK built-in type names, preventing silent ABI/spec misclassification.

---

HAL-005 SOLVED

# Associated type flattening is one-pass and non-recursive (Self::TypeAlias chains)

Low · 2.9 AV:L/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:L

DESCRIPTION

The macro logic that attempts to resolve associated types in impl function signatures performs only a single substitution pass for `Self::Foo` and does not recursively resolve chained aliases. This can yield incorrect spec/wrapper generation or confusing compile failures for contracts that use associated types in public interfaces.

## Code Location

[data-start="309"]

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk-macros/src/syn_ext.rs:309-349
fn flatten_associated_items_in_impl_fns(imp: &mut ItemImpl) {
    // TODO: Flatten associated consts used in functions.
    // Flatten associated types used in functions.
    let associated_types = imp
        .items
        .iter()
        .filter_map(|item| match item {
            ImplItem::Type(i) => Some((i.ident.clone(), i.ty.clone())),
            _ => None,
        })
        .collect::<HashMap<_, _>>();
    let fn_input_types = imp
        .items
        .iter_mut()
        .filter_map(|item| match item {
            ImplItem::Fn(f) => Some(f.sig.inputs.iter_mut().filter_map(|input| match input {
                FnArg::Typed(t) => Some(&mut t.ty),
                _ => None,
            })),
            _ => None,
        })
        .flatten();
    for t in fn_input_types {
        if let Type::Path(TypePath { qself: None, path }) = t.as_mut() {
            let segments = &path.segments;
            if segments.len() == 2
                && segments.first() == Some(&PathSegment::from(format_ident!("Self")))
            {
                if let Some(PathSegment {
                    arguments: PathArguments::None,
                    ident,
                }) = segments.get(1)
                {
                    if let Some(resolved_ty) = associated_types.get(ident) {
                        *t.as_mut() = resolved_ty.clone();
                    }
                }
            }
        }
    }
}
```

### Impact

- Incorrect or missing spec entries; client generation and cross-contract tooling breakage.

#### RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Implement recursive resolution until no `Self::*` remains.
- Detect unresolved `Self::*` types and emit an actionable compile error.

#### REMEDIATION COMMENT

Addressed in: https://github.com/stellar/rs-soroban-sdk/pull/1780

**SOLVED:** Client remediated this issue by recursively resolving associated type aliases in function signatures and by rejecting unresolved or cyclic cases.

---

HAL-006 RISK ACCEPTED

# Token/SAC event schema fragmentation and legacy publisher exposure (mis-indexing risk)

Low · 2.9 AV:L/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:L

DESCRIPTION

The token ecosystem has multiple overlapping event schemas (same topic, different layouts), and deprecated publishers with legacy topic layouts are still exposed. Event consumers often assume “one topic → one schema”, so multiple live schemas increase the risk of mis-decoding and incorrect off-chain accounting.

## Code Location

[data-start="27"]

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-token-sdk/src/lib.rs:27-29
    pub fn events(&self) -> Events {
        Events::new(&self.0)
    }

// file: rs-soroban-sdk-25.1.0/soroban-token-sdk/src/events.rs:3-11
#[contractevent(topics = ["approve"], data_format = "vec")]
pub struct Approve {
    #[topic]
    pub from: Address,
    #[topic]
    pub spender: Address,
    pub amount: i128,
    pub expiration_ledger: u32,
}

// file: rs-soroban-sdk-25.1.0/soroban-token-sdk/src/events.rs:13-30
#[contractevent(topics = ["transfer"], data_format = "single-value")]
pub struct TransferWithAmountOnly {
    #[topic]
    pub from: Address,
    #[topic]
    pub to: Address,
    pub amount: i128,
}

#[contractevent(topics = ["transfer"], data_format = "map")]
pub struct Transfer {
    #[topic]
    pub from: Address,
    #[topic]
    pub to: Address,
    pub to_muxed_id: Option<u64>,
    pub amount: i128,
}

// file: rs-soroban-sdk-25.1.0/soroban-token-sdk/src/events.rs:39-52
#[contractevent(topics = ["mint"], data_format = "single-value")]
pub struct MintWithAmountOnly {
    #[topic]
    pub to: Address,
    pub amount: i128,
}
```

```rust
#[contractevent(topics = ["mint"], data_format = "map")]
pub struct Mint {
    #[topic]
    pub to: Address,
    pub to_muxed_id: Option<u64>,
    pub amount: i128,
}

// file: rs-soroban-sdk-25.1.0/soroban-token-sdk/src/event.rs:22-38
    #[deprecated = "use soroban_token_sdk::events::Transfer::publish"]
    pub fn transfer(&self, from: Address, to: Address, amount: i128) {
        let topics = (symbol_short!("transfer"), from, to);
        self.env.events().publish(topics, amount);
    }

    #[deprecated = "use soroban_token_sdk::events::Mint::publish"]
    pub fn mint(&self, admin: Address, to: Address, amount: i128) {
        let topics = (symbol_short!("mint"), admin, to);
        self.env.events().publish(topics, amount);
    }

    #[deprecated = "use soroban_token_sdk::events::Clawback::publish"]
    pub fn clawback(&self, admin: Address, from: Address, amount: i128) {
        let topics = (symbol_short!("clawback"), admin, from);
        self.env.events().publish(topics, amount);
    }

// file: rs-soroban-sdk-25.1.0/soroban-sdk/src/token.rs:467-471
    /// # Events
    ///
    /// Emits an event with topics `["clawback", admin: Address, to: Address],
    /// data = amount: i128`
    fn clawback(env: Env, from: Address, amount: i128);
```

## Impact

- Off-chain consumers may drop or misinterpret events, leading to incorrect balances/analytics/compliance views.
- Contracts may unintentionally emit non-canonical legacy event layouts.

### RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Stop exposing deprecated publishers by default and deprecate legacy helpers more aggressively.
- Publish a canonical decoding strategy (topic + value type + map keys) and align docs to canonical schemas.
- Consider explicit schema versioning/discriminators if multiple variants must remain.

### REMEDIATION COMMENT

**RISK ACCEPTED:** Client accepted this risk as an intentional SEP-41 design decision and acknowledged the event schema compatibility trade-off.

---

HAL-007 SOLVED

# build.rs Rust-version guard is ineffective under cross-compilation

Low · 2.9 AV:L/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:L

DESCRIPTION

The SDK attempts to prevent building with unsupported Rust/target combinations using `#[cfg(all(target_family = "wasm", target_os = "unknown"))]` in `build.rs`. However, `build.rs` is compiled and executed for the host, not the target, so this guard may not trigger under cross-compilation.

## Code Location

[data-start="8"]

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk/build.rs:8-13
    #[cfg(all(target_family = "wasm", target_os = "unknown"))]
    if let Ok(version) = rustc_version::version() {
        if version.major == 1 && version.minor >= 82 {
            panic!("Rust compiler 1.82+ with target 'wasm32-unknown-unknown' is unsupported by the Soroban Environment, because the 'wasm32-unknown-unknown' target in Rust 1.82+ has features enabled that are not yet supported and not easily disabled: reference-types, multi-value. Use Rust 1.81 to build for the 'wasm32-unknown-unknown' target.");
        }
    }
```

## Impact

- Policy enforcement can fail open: CI may produce WASM with toolchain features the runtime does not support.

RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Check the target triple using Cargo-provided env vars (e.g., `TARGET`, `CARGO_CFG_TARGET_*`) rather than `#[cfg(...)]` in `build.rs`.
- Fail with a clear diagnostic when the target/toolchain combination is unsupported.

REMEDIATION COMMENT

Addressed in: https://github.com/stellar/rs-soroban-sdk/pull/1771

**SOLVED:** Client remediated this issue by using target-aware environment checks in build.rs so unsupported Rust and wasm target combinations are caught during cross-compilation.

HAL-008 NOT APPLICABLE

# Macro parsing discards outer attributes on trait/impl inputs (#[cfg] not propagated)

Low · 2.9 AV:L/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:L

DESCRIPTION

Some macros parse items using a wrapper that consumes and discards outer attributes before parsing the underlying `trait` / `impl`. As a result, attributes like `#[cfg(...)]` on the input item may not apply to generated code.

## Code Location

[data-start="188"]

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk-macros/src/syn_ext.rs:188-203
impl Parse for HasFnsItem {
    fn parse(input: ParseStream) -> syn::Result<Self> {
        _ = input.call(Attribute::parse_outer);
        _ = input.parse::<Token![pub]>();
        let lookahead = input.lookahead1();
        if lookahead.peek(Token![trait]) {
            let t = input.parse()?;
            Ok(HasFnsItem::Trait(t))
        } else if lookahead.peek(Token![impl]) {
            let mut imp = input.parse()?;
            flatten_associated_items_in_impl_fns(&mut imp);
            Ok(HasFnsItem::Impl(imp))
        } else {
            Err(lookahead.error())
        }
    }
```

## Impact

- Build/DX footgun: cfg-gated impls/traits can still produce unconditional generated code, causing compilation failures or confusing “missing symbol” errors.

RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Preserve and re-emit outer attributes (or wrap all derived output in the same attributes) so cfg-gating behaves as users expect.

REMEDIATION COMMENT

**NOT APPLICABLE:** The client remediation attached to this finding references PR 1771, which fixes the separate build.rs cross-compilation check finding. No applicable remediation for this macro outer-attribute propagation issue was identified, so this item should not be marked solved based on that PR.

HAL-009 SOLVED

# contractevent(data_format="single-value") does not enforce single-field and can generate invalid Rust

Low · 2.9 AV:L/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:L

DESCRIPTION

In the `single-value` event data path, the macro expands the data expression using a repetition with no separator. If more than one data field exists, the expansion produces invalid tokens and confusing compiler errors rather than an actionable diagnostic.

## Code Location

[data-start="238"]

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk-macros/src/derive_event.rs:238-245
    let data_to_val = match args.data_format {
        DataFormat::SingleValue if data_params_count == 0 => quote! {
            #path::Val::VOID.to_val()
        },
        DataFormat::SingleValue => quote! {
            use #path::IntoVal;
            #(self.#data_idents.into_val(env))*
        },
```

## Impact

- Build-time failure with poor diagnostics (developer experience + CI friction).

RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Validate that `data_format="single-value"` implies data_fields ∈ {0,1} and emit a clear compile error otherwise.

REMEDIATION COMMENT

Addressed in: https://github.com/stellar/rs-soroban-sdk/pull/1794

**SOLVED:** Client remediated this issue by validating that single-value events have at most one data field and by emitting a clear compile-time error otherwise.

HAL-010 SOLVED

# LedgerSnapshot writes are non-atomic and can truncate outputs on failure

Low · 2.9 AV:L/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:L

## DESCRIPTION

`LedgerSnapshot::write_file` writes directly to the destination path using `File::create`, which truncates the file before serialization completes.
If serialization fails mid-write (disk full, process crash, OOM), this can leave snapshots partially written or corrupted.

### Code Location

[data-start="228"]

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-ledger-snapshot/src/lib.rs:228-237
    /// Write a [`LedgerSnapshot`] to file.
    pub fn write_file(&self, p: impl AsRef<Path>) -> Result<(), Error> {
        let p = p.as_ref();
        if let Some(dir) = p.parent() {
            if !dir.exists() {
                create_dir_all(dir)?;
            }
        }
        self.write(File::create(p)?)
    }
```

### Impact

- Integrity/data-loss risk for snapshot files used in tests/CI tooling when writes fail mid-way.

## RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Write to a temporary file and atomically rename into place on success.
- Consider fsync where appropriate in tooling contexts that require durability guarantees.

## REMEDIATION COMMENT

Addressed in: https://github.com/stellar/rs-soroban-sdk/pull/1796

**SOLVED:** Client remediated this issue by writing snapshots to a temporary file and renaming on success to avoid truncation and partial writes.

HAL-011 SOLVED

# FromXdr guest or WASM error handling can trap before returning Err

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

DESCRIPTION

`FromXdr::from_xdr` returns `Result`, but in guest or Wasm contexts the underlying host deserialization step uses `unwrap_infallible()`, so malformed XDR can still trap before an `Err` is returned.

This is better understood as an API and error-model sharp edge than a hidden parser vulnerability: the `Result` only covers post-deserialization conversion failures, while host deserialization failures follow the guest trap semantics used elsewhere in the SDK.

Callers that expect fully recoverable parsing from untrusted XDR may therefore over-assume error handling in Wasm contracts.

## Code Location

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk/src/xdr.rs:63-66
fn from_xdr(env: &Env, b: &Bytes) -> Result<Self, Self::Error> {
    let t = env.deserialize_from_bytes(b.into()).unwrap_infallible();
    T::try_from_val(env, &t)
}
```

## Impact

- Guest contracts parsing attacker-controlled XDR can trap rather than handling an `Err`, making the issue primarily an API-expectation and robustness concern.

RECOMMENDATION

- Clarify the guest versus native error-model distinction in the `FromXdr` documentation.
- If desired, provide a more explicitly named API or documentation example that makes the trapping behavior obvious in Wasm contexts.

REMEDIATION COMMENT

Addressed in: https://github.com/stellar/rs-soroban-sdk/pull/1767

**SOLVED:** Client remediated this issue by clarifying the guest versus native FromXdr error model so trap behavior is explicit in the documentation.

HAL-012 RISK ACCEPTED

# LedgerSnapshot uses linear scans for lookup and update (large-snapshot performance footgun)

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

DESCRIPTION

`LedgerSnapshot` stores entries in a `Vec` and performs lookups with linear scans and updates with repeated linear `position` checks.
This is a real scalability characteristic, but it is best understood as an off-chain tooling and test-harness performance footgun rather than a direct security vulnerability. The issue becomes relevant when unusually large snapshots are processed in CI, tooling, or local automation.

## Code Location

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-ledger-snapshot/src/lib.rs:192-208
for (k, e) in entries {
    let i = self.ledger_entries.iter().position(|(ik, _)| **ik == **k);
    // ...
}

// file: rs-soroban-sdk-25.1.0/soroban-ledger-snapshot/src/lib.rs:260-264
match self.ledger_entries.iter().find(|(k, _)| **k == **key) {
    Some((_, v)) => Ok(Some((Rc::new(*v.0.clone()), v.1))),
    None => Ok(None),
}
```

## Impact

- Large snapshots can cause avoidable CPU overhead in off-chain tooling and test environments.

RECOMMENDATION

- Document the intended snapshot scale and consider map-based indexing if larger datasets are expected.
- If the current structure is retained, document the performance trade-off explicitly for tooling authors.

REMEDIATION COMMENT

**RISK ACCEPTED:** Client accepted this tooling performance risk, noting that ledger snapshots are intended as test-only tooling where correctness is prioritized over performance and that optimization can be revisited if legitimate test cases are impacted.

HAL-013 SOLVED

# soroban-spec-rust generator assumes trusted spec identifiers when generating bindings

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

DESCRIPTION

`soroban-spec-rust` generates Rust identifiers from spec names using `to_utf8_string().unwrap()` and `format_ident!`.
This is a real robustness limitation when a tool intentionally generates bindings from arbitrary or untrusted Wasm specifications. The behavior is narrower than a general SDK vulnerability: it affects off-chain binding/codegen workflows that ingest attacker-controlled specs, not normal contract execution.
The generator therefore assumes trusted or compiler-produced spec identifiers and can panic when used as an ingestion pipeline for malformed external specs.

## Code Location

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-spec-rust/src/types.rs:15-23
// TODO: Replace the unwrap()s in this code with returning Result.
// TODO: Create Idents in a way that we can get a Result back and return it too
// because at the moment the format_ident! calls can panic if the inputs do not
// result in a valid ident.
pub fn generate_struct(spec: &ScSpecUdtStructV0) -> TokenStream {
    let ident = format_ident!("{}", spec.name.to_utf8_string().unwrap());
}
```

## Impact

- Off-chain binding/codegen tooling that ingests arbitrary contract specs can crash on malformed identifier data.

RECOMMENDATION

- Return structured errors instead of panicking when spec names are not valid UTF-8 or valid Rust identifiers.
- Document clearly that the current generator assumes trusted or compiler-produced specs.

REMEDIATION COMMENT

Addressed in: https://github.com/stellar/rs-soroban-sdk/pull/1810

**SOLVED:** Client successfully remediated this issue by making the soroban-spec-rust code-generation path fallible, returning structured errors instead of panicking on malformed or invalid spec identifiers.

HAL-014 NOT APPLICABLE

# Cross-contract call helpers favor documented panics and coarse error reporting over recoverability

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

## DESCRIPTION

The typed cross-contract call helpers deliberately trade recoverability for convenience. `invoke_contract<T>` documents that it can panic if the return value cannot be converted into `T`, and `try_invoke_contract` uses an error-shaped-value heuristic that can be coarse in edge cases.

This is better framed as an API ergonomics and diagnostics issue than as a hidden on-chain vulnerability. The behavior is real, but it is documented and primarily affects how integrators reason about error handling and return conversion.

### Code Location

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk/src/env.rs:382-403
/// Will panic if the value returned from the contract cannot be converted
/// into the type `T`.
pub fn invoke_contract<T>(...) -> T {
    // ...
    T::try_from_val(self, &rv).map_err(|_| ConversionError).unwrap()
}

// file: rs-soroban-sdk-25.1.0/soroban-sdk/src/env.rs:417-428
match internal::Error::try_from_val(self, &rv) {
    Ok(err) => Err(E::try_from(err).map_err(Into::into)),
    Err(ConversionError) => Ok(T::try_from_val(self, &rv)),
}
```

### Impact

- Integrators can over-assume recoverability or diagnostic fidelity when composing cross-contract calls.

## RECOMMENDATION

- Keep the documentation explicit about the panic and error-classification trade-offs.
- If desired, consider adding APIs with more explicit names or richer return typing for callers that need stronger recoverability guarantees.

## REMEDIATION COMMENT

**NOT APPLICABLE:** Client clarified that this is an API ergonomics preference rather than a security finding, since callers that need recoverable cross-contract errors can use try_invoke_contract.

HAL-015 SOLVED

# testutils auth helpers can mask authorization bugs (mock_all_auths, authorize_as_current_contract)

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

DESCRIPTION

The test environment provides powerful helpers that intentionally diverge from production authorization semantics (e.g., `mock_all_auths` and invoker-authorization helpers). When used indiscriminately, these helpers can over-authorize call trees and hide missing or mis-scoped `require_auth` checks.

## Code Location

[data-start="1117"]

RUST

```
// file: rs-soroban-sdk-25.1.0/soroban-sdk/src/env.rs:1117-1300
    /// Set authorizations and signatures in the environment which will be
    /// consumed by contracts when they invoke [`Address::require_auth`] or
    /// [`Address::require_auth_for_args`] functions.
    ///
    /// Requires valid signatures for the authorization to be successful.
    ///
    /// This function can also be called on contract clients.
    ///
    /// To mock auth for testing, without requiring valid signatures, use
    /// [`mock_all_auths`][Self::mock_all_auths] or
    /// [`mock_auths`][Self::mock_auths]. If mocking of auths is enabled,
    /// calling [`set_auths`][Self::set_auths] disables any mocking.
    pub fn set_auths(&self, auths: &[SorobanAuthorizationEntry]) {
        self.env_impl
            .set_authorization_entries(auths.to_vec())
            .unwrap();
    }

    /// Mock authorizations in the environment which will cause matching invokes
    /// of [`Address::require_auth`] and [`Address::require_auth_for_args`] to
    /// pass.
    ///
    /// This function can also be called on contract clients.
    ///
    /// Authorizations not matching a mocked auth will fail.
    ///
    /// To mock all auths, use [`mock_all_auths`][Self::mock_all_auths].
    ///
    /// ### Examples
    /// ```
    /// use soroban_sdk::{contract, contractimpl, Env, Address, testutils::{Address as _,
MockAuth, MockAuthInvoke}, IntoVal};
    ///
    /// #[contract]
    /// pub struct HelloContract;
    ///
    /// #[contractimpl]
    /// impl HelloContract {
    ///     pub fn hello(env: Env, from: Address) {
    ///         from.require_auth();
    ///         // TODO
    ///     }
    /// }
```

```
    ///
    /// #[test]
    /// fn test() {
    /// # }
    /// # fn main() {
    ///     let env = Env::default();
    ///     let contract_id = env.register(HelloContract, ());
    ///
    ///     let client = HelloContractClient::new(&env, &contract_id);
    ///     let addr = Address::generate(&env);
    ///     client.mock_auths(&[
    ///         MockAuth {
    ///             address: &addr,
    ///             invoke: &MockAuthInvoke {
    ///                 contract: &contract_id,
    ///                 fn_name: "hello",
    ///                 args: (&addr,).into_val(&env),
    ///                 sub_invokes: &[],
// ...
    /// }
    /// ```
    pub fn mock_all_auths(&self) {
        self.env_impl.switch_to_recording_auth(true).unwrap();
    }

    /// A version of `mock_all_auths` that allows authorizations that are not
    /// present in the root invocation.
    ///
    /// Refer to `mock_all_auths` documentation for general information and
    /// prefer using `mock_all_auths` unless non-root authorization is required.
    ///
    /// The only difference from `mock_all_auths` is that this won't return an
    /// error when `require_auth` hasn't been called in the root invocation for
    /// any given address. This is useful to test contracts that bundle calls to
    /// another contract without atomicity requirements (i.e. any contract call
    /// can be frontrun).
    ///
    /// ### Examples
    /// ```
    /// use soroban_sdk::{contract, contractimpl, Env, Address, testutils::Address as _};
    ///
    /// #[contract]
    /// pub struct ContractA;
    ///
    /// #[contractimpl]
    /// impl ContractA {
    ///     pub fn do_auth(env: Env, addr: Address) {
    ///         addr.require_auth();
    ///     }
    /// }
    /// #[contract]
    /// pub struct ContractB;
    ///
    /// #[contractimpl]
    /// impl ContractB {
    ///     pub fn call_a(env: Env, contract_a: Address, addr: Address) {
    ///         // Notice there is no `require_auth` call here.
    ///         ContractAClient::new(&env, &contract_a).do_auth(&addr);
    ///     }
    /// }
    /// #[test]
    /// fn test() {
    /// # }
    /// # fn main() {
    ///     let env = Env::default();
    ///     let contract_a = env.register(ContractA, ());
    ///     let contract_b = env.register(ContractB, ());
    ///     // The regular `env.mock_all_auths()` would result in the call
    ///     // failure.
    ///     env.mock_all_auths_allowing_non_root_auth();
    ///
```

```
    ///     let client = ContractBClient::new(&env, &contract_b);
    ///     let addr = Address::generate(&env);
    ///     client.call_a(&contract_a, &addr);
    /// }
    /// ```
    pub fn mock_all_auths_allowing_non_root_auth(&self) {
        self.env_impl.switch_to_recording_auth(false).unwrap();
    }

// file: rs-soroban-sdk-25.1.0/soroban-sdk/src/env.rs:431-449
    /// Authorizes sub-contract calls on behalf of the current contract.
    ///
    /// All the direct calls that the current contract performs are always
    /// considered to have been authorized. This is only needed to authorize
    /// deeper calls that originate from the next contract call from the current
    /// contract.
    ///
    /// For example, if the contract A calls contract B, contract
    /// B calls contract C and contract C calls `A.require_auth()`, then an
    /// entry corresponding to C call has to be passed in `auth_entries`. It
    /// doesn't matter if contract B called `require_auth` or not. If contract A
    /// calls contract B again, then `authorize_as_current_contract` has to be
    /// called again with the respective entries.
    ///
    ///
    pub fn authorize_as_current_contract(&self, auth_entries: Vec<InvokerContractAuthEntry>)
{
        internal::Env::authorize_as_curr_contract(self, auth_entries.to_object())
            .unwrap_infallible();
    }
```

## Impact

- False confidence: contracts can pass CI while missing critical authorization checks.
- Authorization scope mistakes can go undetected under permissive mocks.

### RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Prefer `set_auths` with realistic authorization entries when testing auth-sensitive logic.
- Add explicit negative tests and verify authorization trees using `env.auths()`.
- Strengthen docs to clearly label these helpers as semantic gaps from production.

### REMEDIATION COMMENT

Addressed in: https://github.com/stellar/rs-soroban-sdk/pull/1798

**SOLVED:** Client remediated this issue by expanding the testutils authorization documentation and examples to warn about auth-mocking gaps and to verify authorization trees explicitly.

---

HAL-016 SOLVED

# Typed collection convenience iterators unwrap conversion failures in raw-Val and storage-driven flows

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

DESCRIPTION

Typed collection convenience APIs such as default iterators and key or value extraction paths use unwrap-based conversion internally.
The behavior is real, but the broadest framing is too strong: standard contract entrypoint arguments usually benefit from host-side interface validation. The sharper risk is in raw-`Val`, storage-driven, or dynamically composed flows where a collection can contain values that do not actually match the expected generic type.

## Code Location

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk/src/iter.rs:34-44
impl<T, I> Iterator for UnwrappedIter<I>
where
    I: Iterator<Item = Result<T, ConversionError>>,
{
    fn next(&mut self) -> Option<Self::Item> {
        self.iter.next().map(Result::unwrap)
    }
}
```

## Impact

- Storage-driven or manually composed typed collections can trap when convenience iteration assumes successful element conversion.

RECOMMENDATION

- Prefer `try_iter` and other fallible APIs when collection contents may originate from storage or raw values.
- Clarify in docs that the convenience iterators assume well-typed contents.

REMEDIATION COMMENT

Addressed in: https://github.com/stellar/rs-soroban-sdk/pull/1774

**SOLVED:** Client remediated this issue by documenting the panic behavior of convenience iterators and by directing callers to use try_iter when conversion can fail.

HAL-017 SOLVED

# Crypto scalar wrapper construction does not enforce canonical encoding invariants

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

## DESCRIPTION

Some crypto scalar wrapper constructors accept raw byte or `U256` inputs without enforcing canonical encoding or range invariants at the SDK wrapper layer.
This is narrower than a direct malleability vulnerability in every use case. The practical impact depends on downstream protocols that serialize, compare, or hash these wrappers directly and assume a single canonical byte representation.
The issue is therefore best understood as an API and protocol-assumption sharp edge rather than a universally exploitable on-chain defect.

### Code Location

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk/src/crypto/bls12_381.rs:391-401
impl From<U256> for Fr {
    fn from(value: U256) -> Self {
        Self(value)
    }
}
```

### Impact

- Downstream protocols can over-assume canonical scalar encodings when using SDK wrapper types directly.

## RECOMMENDATION

- Document explicitly which constructors preserve raw encodings versus enforce canonical field or scalar invariants.
- Consider adding checked constructors for callers that need canonical encodings by construction.

## REMEDIATION COMMENT

Addressed in: https://github.com/stellar/rs-soroban-sdk/security/advisories/GHSA-x2hw-px52-wp4m

**SOLVED:** Client remediated this issue by canonicalizing Fr construction paths and by validating affected field inputs in the patched advisory release.

HAL-018 NOT APPLICABLE

# Token metadata helpers assume initialized state and do not validate metadata values

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

DESCRIPTION

The token metadata helpers assume metadata has already been initialized and read it with unwrap-based access. The metadata values themselves are also left unconstrained at the SDK helper layer.
This is better framed as an API ergonomics and validation-gap issue than as a standalone security vulnerability. The main sharp edges are trapping on unset state and leaving downstream integrators to define their own constraints for fields such as symbol and name.

## Code Location

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-token-sdk/src/metadata.rs:26-33
pub fn get_metadata(e: &Env) -> TokenMetadata {
    e.storage()
        .instance()
        .get(&METADATA_KEY)
        .unwrap_optimized()
        .unwrap_optimized()
}
```

## Impact

- Unset metadata can trap, and metadata quality constraints are left to downstream contracts or integrators.

RECOMMENDATION

- Document the initialization assumption explicitly.
- Consider providing a fallible getter or helper guidance for callers that need optional metadata behavior or tighter field validation.

REMEDIATION COMMENT

**NOT APPLICABLE:** Client clarified that this is a storage interaction and validation guidance item rather than a security finding, as contracts are expected not to allow malformed token metadata to be written to ledger state.

HAL-019 NOT APPLICABLE

# contractimport! does not require SHA-256 pinning and accepts arbitrary paths (integrity/reproducibility footgun)

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

DESCRIPTION

`contractimport!` embeds WASM bytes from a filesystem path at compile time. Unlike hardened workflows, SHA-256 verification is optional and only applied if explicitly provided.
Path resolution accepts absolute paths and relative paths without canonicalization, allowing `..` segments. It also panics if `CARGO_MANIFEST_DIR` is missing, producing a panic rather than a compile error.

## Code Location

[data-start="7"]

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk-macros/src/path.rs:7-17
pub fn abs_from_rel_to_manifest(path: impl Into<PathBuf>) -> PathBuf {
    let path: PathBuf = path.into();
    if path.is_relative() {
        let root: PathBuf = env::var("CARGO_MANIFEST_DIR")
            .expect("CARGO_MANIFEST_DIR environment variable is required to be set")
            .into();
        root.join(path)
    } else {
        path
    }
}

// file: rs-soroban-sdk-25.1.0/soroban-sdk-macros/src/lib.rs:652-690
#[derive(Debug, FromMeta)]
struct ContractImportArgs {
    file: String,
    #[darling(default)]
    sha256: darling::util::SpannedValue<Option<String>>,
}
#[proc_macro]
pub fn contractimport(metadata: TokenStream) -> TokenStream {
    let args = match NestedMeta::parse_meta_list(metadata.into()) {
        Ok(v) => v,
        Err(e) => {
            return TokenStream::from(darling::Error::from(e).write_errors());
        }
    };
    let args = match ContractImportArgs::from_list(&args) {
        Ok(v) => v,
        Err(e) => return e.write_errors().into(),
    };

    // Read WASM from file.
    let file_abs = path::abs_from_rel_to_manifest(&args.file);
    let wasm = match fs::read(file_abs) {
        Ok(wasm) => wasm,
        Err(e) => {
            return Error::new(Span::call_site(), e.to_string())
                .into_compile_error()
```

```
                .into()
        }
    };

    // Generate.
    match generate_from_wasm(&wasm, &args.file, args.sha256.as_deref()) {
        Ok(code) => quote! { #code },
        Err(e @ GenerateFromFileError::VerifySha256 { .. }) => {
            Error::new(args.sha256.span(), e.to_string()).into_compile_error()
        }
        Err(e) => Error::new(Span::call_site(), e.to_string()).into_compile_error(),
    }
    .into()
```

### Impact

- Non-reproducible builds and increased risk of embedding wrong/tampered artifacts in security-sensitive pipelines.

#### RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Provide a hardened mode (feature flag / lint / wrapper macro) that requires SHA-256 pinning.
- Canonicalize paths and optionally enforce “path under manifest dir”.
- Replace `.expect(...)` with compile-time diagnostics instead of panicking.

#### REMEDIATION COMMENT

**NOT APPLICABLE:** After review, this reflects an expected developer-controlled compile-time workflow choice rather than a security issue. Optional SHA-256 verification for `contractimport!` and manifest-relative path resolution are acceptable here, and the documentation update addresses the main gap.

---

HAL-020 SOLVED

# BytesN convenience APIs have edge-case semantics and performance footguns

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

## DESCRIPTION

`BytesN` includes convenience APIs whose behavior can be surprising at the edges. In particular, `is_empty()` is hard-coded to `false`, and some array-conversion paths iterate byte-by-byte instead of using the bulk copy helper.

This is best understood as a correctness and performance footgun rather than a security issue. The main practical risks are surprising semantics for rare `N == 0` cases and avoidable cost in hot paths that convert through the slower convenience conversions.

### Code Location

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk/src/bytes.rs:1229-1231
pub fn is_empty(&self) -> bool {
    false
}
```

### Impact

- Developers can encounter surprising edge-case semantics or pay avoidable conversion costs.

## RECOMMENDATION

- Document the fixed-length semantics of `BytesN` explicitly, especially for `N == 0`.
- Recommend bulk-copy APIs such as `to_array` where performance matters.

## REMEDIATION COMMENT

Addressed in: https://github.com/stellar/rs-soroban-sdk/pull/1733

**SOLVED:** Client remediated this issue by correcting BytesN::is_empty for zero-length BytesN values and by adding coverage for zero and nonzero lengths.

HAL-021 NOT APPLICABLE

# Macro-generated identifiers and helper names can collide across generated surfaces

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

DESCRIPTION

The macro system derives several generated identifiers and helper names from type names or last path segments. In larger crates or unusual macro compositions, these generated names can collide. The collision issue is real, but the broader "leak internal surface area" framing is overstated. This is best described as a code-generation and developer-experience hazard that can produce duplicate definitions or confusing generated surfaces at build time.

## Code Location

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk-macros/src/lib.rs:225-247
// It would be ideal to include the trait ident in the args and client names,
// but we're only currently generating those based on the type that is
// implementing the trait. This could cause conflicts if someone is using
// more than one trait for the same type.
let args_ident = format!("{}Args", ident);
let client_ident = format!("{}Client", ident);
```

## Impact

- Generated names can collide and cause build failures or confusing generated code in multi-trait or multi-type macro usage patterns.

RECOMMENDATION

- Include more context, such as trait identifiers, in generated helper names where feasible.
- Document the collision caveat for projects using multiple trait-driven code-generation paths.

REMEDIATION COMMENT

**NOT APPLICABLE:** Client requested additional evidence for unexpected function exposure or overwrite behavior. Based on review, same-name generated contract functions collide at build time rather than exposing an exploitable runtime surface.

HAL-022 NOT APPLICABLE

# testutils function registry uses global mutable state and panics on poisoned mutex

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

## DESCRIPTION

When `feature="testutils"` is enabled, the generated contract function registry uses a global `Mutex` and calls `.lock().unwrap()`. If any test panics while holding the lock (poisoning the mutex), subsequent tests may panic immediately.

### Code Location

[data-start="157"]

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk-macros/src/lib.rs:157-177
    if cfg!(feature = "testutils") {
        output.extend(quote! {
            mod #fn_set_registry_ident {
                use super::*;

                extern crate std;
                use std::sync::Mutex;
                use std::collections::BTreeMap;

                pub type F = #crate_path::testutils::ContractFunctionF;

                static FUNCS: Mutex<BTreeMap<&'static str, &'static F>> =
Mutex::new(BTreeMap::new());

                pub fn register(name: &'static str, func: &'static F) {
                    FUNCS.lock().unwrap().insert(name, func);
                }

                pub fn call(name: &str, env: #crate_path::Env, args: &[#crate_path::Val]) ->
Option<#crate_path::Val> {
                    let fopt: Option<&'static F> = FUNCS.lock().unwrap().get(name).map(|f|
f.clone());
                    fopt.map(|f| f(env, args))
                }
```

### Impact

- Flaky tests and cascading failures after a single panic in test execution.

## RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Handle poisoned mutexes explicitly (recover inner state) or use a poison-tolerant approach for test-only registries.

REMEDIATION COMMENT

**NOT APPLICABLE:** After review, no contract or test function executes while this mutex is held, so poisoning would require a panic in the registry bookkeeping itself and is not a meaningful issue.

HAL-023 NOT APPLICABLE

# Debug/test-only helpers can panic unexpectedly (Logs::add, native String formatting)

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

DESCRIPTION

Some debug/native/test-only helper implementations use `unwrap()`/panics:

- `Logs::add` calls `env.log_from_slice(...).unwrap()` under `cfg!(debug_assertions)`.
- Native `String` formatting converts through XDR and calls `to_utf8_string().unwrap()`.

These do not typically affect production WASM builds, but can cause surprising crashes in CI or local debug workflows.

## Code Location

[data-start="124"]

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk/src/logs.rs:124-134
    pub fn add(&self, msg: &'static str, args: &[Val]) {
        if cfg!(debug_assertions) {
            let env = self.env();
            env.log_from_slice(msg, args).unwrap();

            #[cfg(any(test, feature = "testutils"))]
            {
                use crate::testutils::Logs;
                std::println!("{}", self.all().last().unwrap());
            }
        }

// file: rs-soroban-sdk-25.1.0/soroban-sdk/src/string.rs:203-213
#[cfg(not(target_family = "wasm"))]
impl core::fmt::Display for String {
    fn fmt(&self, f: &mut core::fmt::Formatter<'_>) -> Result<(), core::fmt::Error> {
        let sc_val: ScVal = self.try_into().unwrap();
        if let ScVal::String(ScString(s)) = sc_val {
            let utf8_s = s.to_utf8_string().unwrap();
            write!(f, "{utf8_s}")?;
        } else {
            panic!("value is not a string");
        }
        Ok(())
```

## Impact

- Reduced test/CI robustness; failures surface as panics rather than actionable diagnostics.

RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Prefer propagating/logging errors rather than unwrapping in debug helpers (or include actionable panic messages).
- Keep fallible formatting paths for native/test use.

REMEDIATION COMMENT

**NOT APPLICABLE:** After review, these are debug or native developer-facing behaviors rather than meaningful security issues in production contract execution.

---

HAL-024 NOT APPLICABLE

# Panic-oriented parsing helpers can trap on invalid caller-supplied strings (Address::from_str, MuxedAddress::from_str, Symbol::new)

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

DESCRIPTION

Several convenience helpers intentionally panic or trap on invalid contents instead of returning a recoverable error:

- `Address::from_str` and `MuxedAddress::from_str` abort on unsupported or malformed strkey contents.
- `Symbol::new` aborts on invalid characters or oversized input.

This can be reached when a contract accepts caller-controlled `String` or `Bytes` values and then parses them internally after ABI decoding. The issue is narrower than a host-interface vulnerability or a general denial-of-service condition: it is an API sharp edge that can turn malformed user input into an aborting contract path when panic-oriented helpers are used instead of fallible alternatives.

## Code Location

`soroban-sdk/src/address.rs`, `soroban-sdk/src/muxed_address.rs`, `soroban-sdk/src/symbol.rs`

## Impact

- Contracts that parse caller-supplied strings with these helpers can abort instead of returning a structured contract error.

RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Prefer fallible parsing at public contract boundaries when strings or bytes may be caller-controlled.
- Keep documentation and examples explicit that these helpers are panic-oriented convenience APIs.

REMEDIATION COMMENT

**NOT APPLICABLE:** Client clarified that this is a feature request for additional fallible parsing convenience APIs rather than a security finding, and the current helpers are panic-oriented convenience functions.

HAL-025 SOLVED

# Storage TTL helpers assume host invariants and trap on invalid parameter flows

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

## DESCRIPTION

The storage TTL helper APIs assume that host-side ledger invariants hold and propagate host failures through trap-oriented paths such as `unwrap_infallible()`.
This is a real sharp edge, but it is best treated as an API and documentation issue rather than a direct vulnerability. The underflow and invalid-parameter cases primarily matter in misconfigured, test, or otherwise abnormal flows where callers over-assume graceful recoverability.

### Code Location

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk/src/storage.rs:153-157
pub fn max_ttl(&self) -> u32 {
    let max = self.storage.env.max_live_until_ledger().unwrap_infallible();
    let seq = self.storage.env.ledger().sequence();
    max - seq
}
```

### Impact

- Callers can encounter surprising traps or edge-case arithmetic behavior when host invariants do not hold as assumed.

## RECOMMENDATION

- Document the host invariant assumptions behind these helpers.
- Consider defensive checked or saturating behavior where that better matches caller expectations.

## REMEDIATION COMMENT

Addressed in: https://github.com/stellar/rs-soroban-sdk/pull/1792

**SOLVED:** Client remediated this issue by using saturating arithmetic in max_ttl to avoid underflow in misconfigured or abnormal flows.

HAL-026 SOLVED

# register_* test utilities are not panic-safe and can leave Env state inconsistent

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

DESCRIPTION

Several test-only helpers snapshot and replace the host auth manager, then perform actions that can trap/panic. If an intermediate step aborts, restoration code may not run, leaving the environment in a confusing state for subsequent test steps.
Related issues include fixed all-zero salts in registration and semantics that diverge from on-chain upgrades.

## Code Location

[data-start="1028"]

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk/src/env.rs:1028-1045
        let token_id: Address = self
            .env_impl
            .invoke_function(create)
            .unwrap()
            .try_into_val(self)
            .unwrap();

        let prev_auth_manager = self.env_impl.snapshot_auth_manager().unwrap();
        self.env_impl
            .switch_to_recording_auth_inherited_from_snapshot(&prev_auth_manager)
            .unwrap();
        self.invoke_contract::<()>(
            &token_id,
            &soroban_sdk_macros::internal_symbol_short!("set_admin"),
            (admin,).try_into_val(self).unwrap(),
        );
        self.env_impl.set_auth_manager(prev_auth_manager).unwrap();


// file: rs-soroban-sdk-25.1.0/soroban-sdk/src/env.rs:1086-1113
        let prev_auth_manager = self.env_impl.snapshot_auth_manager().unwrap();
        self.env_impl
            .switch_to_recording_auth_inherited_from_snapshot(&prev_auth_manager)
            .unwrap();
        let args_vec: std::vec::Vec<xdr::ScVal> =
            constructor_args.iter().map(|v| v.into_val(self)).collect();
        let contract_id: Address = self
            .env_impl
            .invoke_function(xdr::HostFunction::CreateContractV2(
                xdr::CreateContractArgsV2 {
                    contract_id_preimage: xdr::ContractIdPreimage::Address(
                        xdr::ContractIdPreimageFromAddress {
                            address: xdr::ScAddress::Contract(xdr::ContractId(xdr::Hash(
                                self.with_generator(|mut g| g.address()),
                            ))),
                            salt: xdr::Uint256([0; 32]),
                        },
                    ),
                    executable,
                    constructor_args: args_vec.try_into().unwrap(),
                },
```

```
            ))
            .unwrap()
            .try_into_val(self)
            .unwrap();

        self.env_impl.set_auth_manager(prev_auth_manager).unwrap();
```

## Impact

- Flaky tests and misleading results when one failing call corrupts the test environment for later assertions.

RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Use guard patterns (Drop-based restoration) so state is restored even on panics where feasible.
- Use randomized salts for test deployments to reduce collision risk.
- Document test-only semantics and divergences from on-chain behavior clearly.

REMEDIATION COMMENT

Addressed in: https://github.com/stellar/rs-soroban-sdk/pull/1803

**SOLVED:** Client remediated this issue by restoring auth-manager state across failing register flows and by adding a regression test for panic safety.

---

HAL-027 ACKNOWLEDGED

# WASM bump allocator is intentionally non-freeing (memory-growth footgun inside an invocation)

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

## DESCRIPTION

For WASM builds with `feature="alloc"`, the SDK provides a minimal bump allocator whose `dealloc` is a no-op. This aligns with a single-invocation model but means dropping allocations does not reclaim linear memory within an invocation.

## Code Location

[data-start="68"]

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk/src/lib.rs:68-71
// Here we provide a `#[global_allocator]` that is a minimal non-freeing bump
// allocator, appropriate for a WASM blob that runs a single contract call.
#[cfg(all(feature = "alloc", target_family = "wasm"))]
mod alloc;

// file: rs-soroban-sdk-25.1.0/soroban-sdk/src/alloc.rs:21-105
static mut LOCAL_ALLOCATOR: BumpPointerLocal = BumpPointerLocal::new();

struct BumpPointerLocal {
    cursor: usize,
    limit: usize,
}

impl BumpPointerLocal {
    const LOG_PAGE_SIZE: usize = 16;
    const PAGE_SIZE: usize = 1 << Self::LOG_PAGE_SIZE; // 64KB
    const MEM: u32 = 0; // Memory 0 is the only legal one currently

    pub const fn new() -> Self {
        Self {
            cursor: 0,
            limit: 0,
        }
    }

    #[inline(always)]
    fn maybe_init_inline(&mut self) {
        if self.limit == 0 {
            // This is a slight over-estimate and ideally we would use __heap_base
            // but that seems not to be easy to access and in any case it is just a
            // convention whereas this is more guaranteed by the wasm spec to work.
            self.cursor = core::arch::wasm32::memory_size(Self::MEM)
                .checked_mul(Self::PAGE_SIZE)
                .unwrap_optimized();
            self.limit = self.cursor;
        }
    }

    #[inline(never)]
    fn maybe_init(&mut self) {
        self.maybe_init_inline()
    }

    // Allocate `bytes` bytes with `align` alignment.
```

```
    #[inline(always)]
    fn alloc(&mut self, bytes: usize, align: usize) -> usize {
        self.maybe_init();
        let start = self
            .cursor
            .checked_next_multiple_of(align)
            .unwrap_optimized();
        let new_cursor = start.checked_add(bytes).unwrap_optimized();
        if new_cursor <= self.limit {
            self.cursor = new_cursor;
            start
        } else {
            self.alloc_slow(bytes, align)
        }
    }

    #[inline(always)]
    fn alloc_slow_inline(&mut self, bytes: usize, align: usize) -> usize {
        let pages = bytes.div_ceil(Self::PAGE_SIZE);
        if core::arch::wasm32::memory_grow(Self::MEM, pages) == usize::MAX {
            core::arch::wasm32::unreachable();
        }
        let bytes_grown = pages.checked_mul(Self::PAGE_SIZE).unwrap_optimized();
        self.limit = self.limit.checked_add(bytes_grown).unwrap_optimized();
        self.alloc(bytes, align)
    }

    #[inline(never)]
    fn alloc_slow(&mut self, bytes: usize, align: usize) -> usize {
        self.alloc_slow_inline(bytes, align)
    }
}

pub struct BumpPointer;

unsafe impl GlobalAlloc for BumpPointer {
    #[inline(always)]
    #[allow(static_mut_refs)]
    unsafe fn alloc(&self, layout: Layout) -> *mut u8 {
        let (bytes, align) = (layout.size(), layout.align());
        let ptr = LOCAL_ALLOCATOR.alloc(bytes, align);
        core::ptr::with_exposed_provenance_mut(ptr)
    }

    #[inline(always)]
    unsafe fn dealloc(&self, _ptr: *mut u8, _layout: Layout) {}
}
```

## Impact

- Budget/memory DoS footgun for contracts that repeatedly allocate large buffers within a single call.

RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Document this prominently as an execution-model constraint and recommend allocation-minimizing patterns.
- Consider an allocator that supports deallocation if the runtime model evolves to reuse WASM instances across calls.

REMEDIATION COMMENT

**ACKNOWLEDGED:** Client acknowledged this informational design trade-off and accepted the documented non-freeing bump allocator behavior for single-invocation WASM execution.

HAL-028 SOLVED

# secp256k1_recover parameter typo (recorvery_id) reduces clarity

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

## DESCRIPTION

The public `Crypto::secp256k1_recover` API spells `recovery_id` as `recorvery_id`. This is a minor papercut that propagates into contract code and reduces clarity when auditing signature recovery flows.

### Code Location

[data-start="155"]

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk/src/crypto.rs:155-168
    /// Recovers the ECDSA secp256k1 public key.
    ///
    /// The public key returned is the SEC-1-encoded ECDSA secp256k1 public key
    /// that produced the 64-byte signature over a given 32-byte message digest,
    /// for a given recovery_id byte.
    pub fn secp256k1_recover(
        &self,
        message_digest: &Hash<32>,
        signature: &BytesN<64>,
        recorvery_id: u32,
    ) -> BytesN<65> {
        let env = self.env();
        CryptoHazmat::new(env).secp256k1_recover(&message_digest.0, signature, recorvery_id)
    }
```

### Impact

- Developer experience/readability degradation; increased audit friction.

## RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Fix the parameter name while preserving backwards compatibility via deprecation if needed.

## REMEDIATION COMMENT

Addressed in: https://github.com/stellar/rs-soroban-sdk/pull/1804

**SOLVED:** Client remediated this issue by renaming the recovery_id parameter for clarity and consistency across the affected crypto interfaces.

HAL-029 NOT APPLICABLE

# AddressPayload constructs XDR manually (hazmat/test feature; fragile encoding assumptions)

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

DESCRIPTION

The `hazmat-address` / test-only `AddressPayload` helper constructs XDR bytes manually by concatenating hardcoded discriminants and raw payload bytes, then calls `Address::from_xdr(...).unwrap_optimized()`.
This is brittle: future XDR layout/discriminant changes require updating “magic bytes”, and mistakes manifest as traps rather than structured errors.

## Code Location

[data-start="49"]

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk/src/address_payload.rs:49-76
    pub fn to_address(&self, env: &Env) -> Address {
        use crate::xdr::FromXdr;
        // Build XDR header and get payload bytes based on payload type:
        let (header, payload_bytes) = match self {
            AddressPayload::AccountIdPublicKeyEd25519(bytes) => (
                [
                    0, 0, 0, 18, // ScVal::Address
                    0, 0, 0, 0, // ScAddress::Account
                    0, 0, 0, 0, // PublicKey::PublicKeyTypeEd25519
                ]
                .as_slice(),
                bytes.as_bytes(),
            ),
            AddressPayload::ContractIdHash(bytes) => (
                [
                    0, 0, 0, 18, // ScVal::Address
                    0, 0, 0, 1, // ScAddress::Contract
                ]
                .as_slice(),
                bytes.as_bytes(),
            ),
        };

        let mut xdr = Bytes::from_slice(env, header);
        xdr.append(payload_bytes);

        Address::from_xdr(env, &xdr).unwrap_optimized()
    }
```

## Impact

- Test/hazmat fragility: subtle encoding mismatches become traps and can be hard to debug.

RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Prefer building XDR via typed `stellar_xdr` structures and serializing them instead of hand-encoding discriminants.

REMEDIATION COMMENT

**NOT APPLICABLE:** Client clarified that this test/hazmat helper is covered by compatibility expectations and tests, and the relevant Stellar XDR discriminants are expected to remain backward-compatible.

---

HAL-030 SOLVED

# Generated WASM export names use a flat method-name namespace

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

## DESCRIPTION

The macro-generated Wasm export names are derived from the method name alone rather than a richer namespace.
This is a real build and linkage limitation, but it is best understood as an unusual multi-contract or duplicate-method-name composition hazard rather than a direct runtime security issue. In practice, collisions typically surface as build or linking failures.

### Code Location

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk-macros/src/derive_fn.rs:139-141
let wrap_export_name = &format!("{}", ident);
#[cfg_attr(target_family = "wasm", export_name = #wrap_export_name)]
```

### Impact

- Unusual multi-contract or duplicate-method-name builds can encounter export-name collisions.

## RECOMMENDATION

- Document the flat export-name behavior explicitly.
- If broader multi-contract support is desired, consider richer export-name namespacing.

## REMEDIATION COMMENT

Addressed in: https://github.com/stellar/rs-soroban-sdk/pull/1809

**SOLVED:** Client remediated this issue by explicitly documenting the flat Wasm export namespace and the resulting collision behavior for contractimpl functions.

HAL-031 SOLVED

# Token migration doc link is incorrect (Address links to MuxedAddress)

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

## DESCRIPTION

The v23 token transfer migration doc defines a link for `Address` that incorrectly points to `MuxedAddress`, which can confuse readers and lead to migration mistakes.

### Code Location

[data-start="78"]

RUST

```
// file: rs-soroban-sdk-25.1.0/soroban-token-sdk/src/_migrating/v23_token_transfer.rs:78-80
//! [`Events::publish`]: crate::events::Events::publish
//! [`Address`]: crate::MuxedAddress
//! [`MuxedAddress`]: crate::MuxedAddress
```

### Impact

- Documentation confusion; increased migration errors.

## RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Fix the link target so `[Address]` links to the correct type.

## REMEDIATION COMMENT

Addressed in: https://github.com/stellar/rs-soroban-sdk/pull/1804

**SOLVED:** Client remediated this issue by correcting the migration guide link to reference Address instead of MuxedAddress.

HAL-032 NOT APPLICABLE

# PRNG is explicitly non-secret and validator-influenceable (misuse breaks security assumptions)

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

DESCRIPTION

The SDK's PRNG is documented as deterministic/predictable and influenced by validators. It must not be used for secret generation or fairness-critical randomness without additional protocol measures.

## Code Location

[data-start="5"]

RUST

```
// file: rs-soroban-sdk-25.1.0/soroban-sdk/src/prng.rs:5-13
//! Do not use the PRNG in this module without a clear understanding of two
//! major limitations in the way it is deployed in the Stellar network:
//!
//!   1. The PRNG is seeded with data that is public as soon as each ledger is
//!      nominated. Therefore it **should never be used to generate secrets**.
//!
//!   2. The PRNG is seeded with data that is under the control of validators.
//!      Therefore it **should only be used in applications where the risk of
//!      validator influence is acceptable**.
```

## Impact

- Contracts that misuse PRNG for secrets/lotteries can be exploited via prediction or validator influence.

RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Use commit–reveal, oracles, or other entropy sources when unpredictability is required (as the docs already suggest).

REMEDIATION COMMENT

**NOT APPLICABLE:** After review, the PRNG limitations are already documented explicitly and this item does not represent a separate vulnerability.

---

HAL-033 SOLVED

# assert_in_contract! is test-only and is a no-op in production builds

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

DESCRIPTION

`assert_in_contract!` expands to a runtime `assert!` only in `test` / `feature="testutils"` builds. In production WASM builds it expands to an empty block.
This is not a vulnerability by itself, but it is important when auditing SDK guard patterns: enforcement may come from host behavior rather than this macro in production.

## Code Location

[data-start="126"]

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk/src/lib.rs:126-142
/// Assert in contract asserts that the contract is currently executing within a
/// contract. The macro maps to code when testutils are enabled or in tests,
/// otherwise maps to nothing.
#[macro_export]
macro_rules! assert_in_contract {
    ($env:expr $(,)?) => {{
        {
            #[cfg(any(test, feature = "testutils"))]
            assert!(
                ($env).in_contract(),
                "this function is not accessible outside of a contract, wrap \
                the call with `env.as_contract()` to access it from a \
                particular contract"
            );
        }
    }};
}
```

## Impact

- Auditors/developers may incorrectly assume a runtime assertion exists in production builds.

RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Keep docs explicit (already are). Consider renaming to make test-only semantics even clearer.

REMEDIATION COMMENT

Addressed in: https://github.com/stellar/rs-soroban-sdk/pull/1806

HAL-034 ACKNOWLEDGED

# Token spec generation hardcodes XDR_LEN and panics at const-eval (brittle maintenance)

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

## DESCRIPTION

`soroban-token-spec` and `stellar-asset-spec` hardcode an `XDR_LEN` constant and use a const-eval panic if the concatenated output size differs. This is a maintenance hazard: harmless schema refactors can cause opaque compile-time panics, and byte-length equality is not a strong correctness guarantee by itself.

### Code Location

[data-start="55"]

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-token-spec/src/lib.rs:55-111
pub(crate) const XDR_INPUT: &[&[u8]] = &[
    &soroban_sdk::token::TokenFnSpec::spec_xdr_allowance(),
    &soroban_sdk::token::TokenFnSpec::spec_xdr_approve(),
    &soroban_sdk::token::TokenFnSpec::spec_xdr_balance(),
    &soroban_sdk::token::TokenFnSpec::spec_xdr_burn(),
    &soroban_sdk::token::TokenFnSpec::spec_xdr_burn_from(),
    &soroban_sdk::token::TokenFnSpec::spec_xdr_decimals(),
    &soroban_sdk::token::TokenFnSpec::spec_xdr_name(),
    &soroban_sdk::token::TokenFnSpec::spec_xdr_symbol(),
    &soroban_sdk::token::TokenFnSpec::spec_xdr_transfer(),
    &soroban_sdk::token::TokenFnSpec::spec_xdr_transfer_from(),
    &soroban_token_sdk::events::Approve::spec_xdr(),
    &soroban_token_sdk::events::TransferWithAmountOnly::spec_xdr(),
    &soroban_token_sdk::events::Transfer::spec_xdr(),
    &TransferWithMuxedString::spec_xdr(),
    &TransferWithMuxedBytes::spec_xdr(),
    &soroban_token_sdk::events::Burn::spec_xdr(),
    &soroban_token_sdk::events::MintWithAmountOnly::spec_xdr(),
    &soroban_token_sdk::events::Mint::spec_xdr(),
    &MintWithMuxedString::spec_xdr(),
    &MintWithMuxedBytes::spec_xdr(),
    &soroban_token_sdk::events::Clawback::spec_xdr(),
];

pub(crate) const XDR_LEN: usize = 6620;

/// Returns the contract spec for a SEP-41 Token contract.
pub const fn xdr() -> &'static [u8] {
    &XDR
}

/// The contract spec for a SEP-41 Token contract.
const XDR: [u8; XDR_LEN] = {
    let input = XDR_INPUT;
    // Concatenate all XDR for each item that makes up the token spec.
    let mut output = [0u8; XDR_LEN];
    let mut input_i = 0;
    let mut output_i = 0;
    while input_i < input.len() {
        let subinput = input[input_i];
        let mut subinput_i = 0;
        while subinput_i < subinput.len() {
            output[output_i] = subinput[subinput_i];
```

```
            output_i += 1;
            subinput_i += 1;
        }
        input_i += 1;
    }

    // Check that the numbers of bytes written is equal to the number of bytes
    // expected in the output.
    if output_i != output.len() {
        panic!("unexpected output length",);
    }

    output
};

// file: rs-soroban-sdk-25.1.0/stellar-asset-spec/src/lib.rs:24-88
pub(crate) const XDR_INPUT: &[&[u8]] = &[
    &soroban_sdk::token::StellarAssetFnSpec::spec_xdr_allowance(),
    &soroban_sdk::token::StellarAssetFnSpec::spec_xdr_authorized(),
    &soroban_sdk::token::StellarAssetFnSpec::spec_xdr_approve(),
    &soroban_sdk::token::StellarAssetFnSpec::spec_xdr_balance(),
    &soroban_sdk::token::StellarAssetFnSpec::spec_xdr_burn(),
    &soroban_sdk::token::StellarAssetFnSpec::spec_xdr_burn_from(),
    &soroban_sdk::token::StellarAssetFnSpec::spec_xdr_clawback(),
    &soroban_sdk::token::StellarAssetFnSpec::spec_xdr_decimals(),
    &soroban_sdk::token::StellarAssetFnSpec::spec_xdr_mint(),
    &soroban_sdk::token::StellarAssetFnSpec::spec_xdr_name(),
    &soroban_sdk::token::StellarAssetFnSpec::spec_xdr_set_admin(),
    &soroban_sdk::token::StellarAssetFnSpec::spec_xdr_admin(),
    &soroban_sdk::token::StellarAssetFnSpec::spec_xdr_set_authorized(),
    &soroban_sdk::token::StellarAssetFnSpec::spec_xdr_symbol(),
    &soroban_sdk::token::StellarAssetFnSpec::spec_xdr_transfer(),
    &soroban_sdk::token::StellarAssetFnSpec::spec_xdr_transfer_from(),
    &soroban_token_sdk::events::Approve::spec_xdr(),
    &soroban_token_sdk::events::TransferWithAmountOnly::spec_xdr(),
    &soroban_token_sdk::events::Transfer::spec_xdr(),
    &soroban_token_spec::TransferWithMuxedString::spec_xdr(),
    &soroban_token_spec::TransferWithMuxedBytes::spec_xdr(),
    &soroban_token_sdk::events::Burn::spec_xdr(),
    &soroban_token_sdk::events::MintWithAmountOnly::spec_xdr(),
    &soroban_token_sdk::events::Mint::spec_xdr(),
    &soroban_token_spec::MintWithMuxedString::spec_xdr(),
    &soroban_token_spec::MintWithMuxedBytes::spec_xdr(),
    &soroban_token_sdk::events::Clawback::spec_xdr(),
    &SetAdmin::spec_xdr(),
    &SetAuthorized::spec_xdr(),
];

pub(crate) const XDR_LEN: usize = 8560;

/// Returns the contract spec for Stellar Asset contract.
pub const fn xdr() -> &'static [u8] {
    &XDR
}

/// The contract spec for the Stellar Asset contract.
const XDR: [u8; XDR_LEN] = {
    let input = XDR_INPUT;
    // Concatenate all XDR for each item that makes up the token spec.
    let mut output = [0u8; XDR_LEN];
    let mut input_i = 0;
    let mut output_i = 0;
    while input_i < input.len() {
        let subinput = input[input_i];
        let mut subinput_i = 0;
        while subinput_i < subinput.len() {
            output[output_i] = subinput[subinput_i];
            output_i += 1;
            subinput_i += 1;
        }
        input_i += 1;
```

```
    }

    // Check that the numbers of bytes written is equal to the number of bytes
    // expected in the output.
    if output_i != output.len() {
        panic!("unexpected output length",);
    }

    output
};
```

## Impact

- Brittle builds and confusing failures when spec inputs change.

RECOMMENDATION

To mitigate this issue, we recommend the following improvements:

- Generate the output length mechanically (or move generation into `build.rs`) and improve diagnostics to report expected vs actual lengths.

REMEDIATION COMMENT

**ACKNOWLEDGED:** Client acknowledged this informational maintenance trade-off and accepted the current hardcoded length guard for token spec generation.

---

HAL-035 NOT APPLICABLE

# soroban-meta parser scales XDR byte limits with input size

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

## DESCRIPTION

`soroban-meta` derives its XDR byte limit from the size of the input metadata buffer itself. This is an accurate parser-budgeting observation, but it is narrower than a generic parsing vulnerability. The practical concern is that tooling ingesting large untrusted metadata blobs grows its parser budget with attacker-controlled input size rather than using a fixed conservative cap.

### Code Location

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-meta/src/read.rs:7-12
pub fn parse_raw(meta: &[u8]) -> Result<Vec<ScMetaEntry>, ReadError> {
    let limits = Limits::len(meta.len());
    Ok(VecM::from_xdr(meta, limits)?.into())
}
```

### Impact

- Off-chain tooling can scale parsing work with attacker-controlled metadata size instead of a fixed conservative bound.

## RECOMMENDATION

- Use fixed or policy-driven parser limits when ingesting metadata from untrusted sources.
- Document that the current helper derives byte limits directly from input size.

## REMEDIATION COMMENT

**NOT APPLICABLE:** After review, this parser operates linearly over already in-memory metadata and does not present a meaningful security issue through its input-size-derived XDR limit.

HAL-036 ACKNOWLEDGED

# LedgerSnapshot::default embeds the current protocol version rather than an obviously empty sentinel

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

DESCRIPTION

`LedgerSnapshot::default()` populates `protocol_version` with the current SDK-era protocol value rather than an obviously empty or placeholder sentinel.
This is better understood as an API-convention and caller-expectation issue than as a vulnerability. The main risk is that tests or tooling may read a partially initialized default snapshot as more meaningful than intended.

## Code Location

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-ledger-snapshot/src/lib.rs:239-244
impl Default for LedgerSnapshot {
    fn default() -> Self {
        Self {
            protocol_version: 25,
            // ...
        }
    }
}
```

## Impact

- Tooling or tests can over-assume semantic meaning from a default snapshot value that is meant only as a convenience initializer.

RECOMMENDATION

- Document the semantics of `Default` clearly.
- If stronger separation is desired, consider an explicitly empty constructor alongside the current convenience default.

REMEDIATION COMMENT

**ACKNOWLEDGED:** Client acknowledged this informational API convention and accepted the default protocol-version initialization aligned with the SDK major version.

HAL-037 NOT APPLICABLE

# LedgerSnapshot::update ignores ledger-info read results in testutils flows

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

## DESCRIPTION

`LedgerSnapshot::update` reads ledger info through a helper and discards the returned `Result` before continuing with entry updates.
This is a real correctness issue in testutils flows, but its scope is narrower than a general vulnerability. The concern is snapshot drift between header metadata and stored entries in failing or unusual test environments rather than production contract execution.

### Code Location

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-ledger-snapshot/src/lib.rs:132-139
pub fn update(&mut self, host: &soroban_env_host::Host) {
    let _result = host.with_ledger_info(|li| {
        self.set_ledger_info(li.clone());
        Ok(())
    });
    self.update_entries(&host.get_stored_entries().unwrap());
}
```

### Impact

- Testutils snapshots can drift from host ledger metadata when ledger-info reads fail but entry updates still proceed.

## RECOMMENDATION

- Propagate or log ledger-info update failures in `update`.
- Document that this helper is intended for test flows and does not currently surface ledger-info read errors.

## REMEDIATION COMMENT

**NOT APPLICABLE:** After review, this testutils-only path behaves acceptably when ledger info is unavailable; retaining the prior snapshot ledger info while updating entries is not a meaningful security issue.

HAL-038 SOLVED

# Generated testutils try_ client functions ignore allow_non_root_auth setting

● Informational · AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

DESCRIPTION

The generated `try_` client methods do not honor the `allow_non_root_auth` testutils setting, even though the non-`try_` client methods do.
This is a real bug in generated test client behavior, but it is limited to testutils-enabled client code and should not be framed as an on-chain production issue. The main consequence is asymmetric auth behavior and false test results when teams rely on the non-root-auth mock setting.

## Code Location

RUST

```rust
// file: rs-soroban-sdk-25.1.0/soroban-sdk-macros/src/derive_client.rs:273-278
if self.mock_all_auths {
    if self.allow_non_root_auth {
        self.env.mock_all_auths_allowing_non_root_auth();
    } else {
        self.env.mock_all_auths();
    }
}

// file: rs-soroban-sdk-25.1.0/soroban-sdk-macros/src/derive_client.rs:306-307
if self.mock_all_auths {
    self.env.mock_all_auths();
}
```

## Impact

- Test client `try_` methods can behave differently from non-`try_` methods under the same testutils auth configuration.

RECOMMENDATION

- Apply the same `allow_non_root_auth` branching logic in generated `try_` methods.
- Document the current asymmetry until the generation logic is aligned.

REMEDIATION COMMENT

Addressed in: https://github.com/stellar/rs-soroban-sdk/pull/1761

**SOLVED:** Client remediated this issue by applying the allow_non_root_auth branching logic to generated try_ client methods and by adding test coverage.

# Disclaimer

*Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.*