OpenZeppelin | security

Stellar Contracts RC v.0.6.0 Audit

January 7, 2026

Table of Contents

Summary

Type	Library
Timeline	From 2025-12-04 To 2025-12-24
Languages	Solidity

Total Issues	10 (9 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	1 (1 resolved)
Medium Severity Issues	3 (2 resolved)
Low Severity Issues	3 (3 resolved)
Notes & Additional Information	3 (3 resolved)

Scope

OpenZeppelin conducted a differential audit of the OpenZeppelin/stellar-contracts repository with HEAD commit 0a89f54 against BASE commit a5fada3.

In scope were the following files:

```
packages/access/src/access_control/mod.rs
packages/access/src/access_control/storage.rs
packages/accounts/src/policies/simple_threshold.rs
packages/accounts/src/policies/spending_limit.rs
packages/accounts/src/policies/weighted_threshold.rs
packages/accounts/src/smart_account/mod.rs
packages/accounts/src/smart_account/storage.rs
packages/accounts/src/verifiers/webauthn.rs
packages/contract-utils/src/math/i128_fixed_point.rs
packages/contract-utils/src/math/i256_fixed_point.rs
packages/contract-utils/src/math/mod.rs
packages/contract-utils/src/math/wad.rs
packages/fee-abstraction/src/lib.rs
packages/fee-abstraction/src/storage.rs
packages/governance/src/lib.rs
packages/governance/src/timelock/mod.rs
packages/governance/src/timelock/storage.rs
packages/macros/src/access_control.rs
packages/macros/src/default_impl_macro.rs
packages/tokens/src/fungible/extensions/allowlist/storage.rs
packages/tokens/src/fungible/extensions/blocklist/storage.rs
packages/tokens/src/fungible/mod.rs
packages/tokens/src/fungible/overrides.rs
packages/tokens/src/fungible/storage.rs
packages/tokens/src/non_fungible/mod.rs
packages/tokens/src/non_fungible/storage.rs
packages/tokens/src/rwa/compliance/storage.rs
packages/tokens/src/rwa/identity_registry_storage/mod.rs
packages/tokens/src/rwa/identity_registry_storage/storage.rs
packages/tokens/src/rwa/storage.rs
```

***Update**: The fixes for the findings highlighted in this report have all been merged at commit b75a2e5.*

Key Changes in Access Control

Role Enumeration and Limits

The storage layer now tracks all existing roles in `ExistingRoles` and enforces `MAX_ROLES = 256`, rejecting new roles with `MaxRolesExceeded` once the limit is reached. Roles are added on first assignment and removed when their last member is revoked.

Admin Transfer Safety and API Normalization

`renounce_admin` now blocks execution while a pending admin transfer exists, returning `TransferInProgress`. Public and macro-generated access-control APIs reorder parameters to consistently take `(account, role, caller)`, and helper checks (`ensure_role`, `ensure_if_admin_or_admin_role`, grant/revoke/renounce) follow the new signature. Macros were updated to match.

Key Changes in Smart Accounts and Policies

Installation Safeguards and Error Namespace

Policy install flows for `simple_threshold`, `weighted_threshold`, and `spending_limit` now reject duplicate installs per `(smart_account, context_rule)` with `AlreadyInstalled`, and their error codes shift into the 31xx/32xx space to avoid collisions. WebAuthn verifier error codes were similarly rebased.

Rolling-Window Accuracy for Spending Limits

The spending-limit window explicitly evicts entries with `ledger_sequence <= current - period`, aligning docs and logic. Cached totals now use the corrected cutoff and short-circuit to `false` when no data exists.

Robust Context Management

Smart accounts expose `get_context_rules_count` for total rule introspection and cap checks. Context rule and policy removal now use `try_uninstall`, allowing cleanup to proceed even if a policy panics during uninstall.

Key Changes in the Math Library

Unified Fixed-Point API with Checked Variants

`SorobanFixedPoint` now lives in `math::mod` with renamed errors (`DivisionByZero`, `Overflow`) and new `checked_fixed_mul_floor/ceil` helpers. Public `muldiv` / `checked_muldiv` wrap common i128 math with explicit rounding control. i128/I256 implementations automatically lift to I256 on overflow and return `Option` in checked paths.

New Wad Type for 18-Decimal Arithmetic

`math::wad::Wad` has been added. It is a fixed-point, 18-decimal type with conversions from integers, token amounts, and prices, plus checked arithmetic (`checked_add/sub/mul/div`, `pow`, token conversion helpers). Operations truncate toward zero and surface overflow through fixed-point errors.

New Fee Abstraction Module

Introduces fee-token allowlisting (optional, count-tracked) with `FeeTokenAlreadyAllowed`, `FeeTokenNotAllowed`, and `TokenCountOverflow` protections. Provides `collect_fee` with lazy or eager allowance, `validate_fee_bounds`, and rich events (`FeeCollected`, `ForwardExecuted`, `TokensSwept`). `auth_user_and_invoke` forwards calls after auth, and `sweep_token` safely drains accumulated balances (`NoTokensToSweep`).

New Governance Timelock Module

Adds a full timelock utility: operations are hashed (target, fn, args, predecessor, salt) via Keccak, scheduled with per-op delay ≥ `min_delay`, and tracked through `OperationState` (Unset/Waiting/Ready/Done). Execution validates readiness and predecessor completion. The module now also exposes state queries (`get_operation_state`, `is_operation_ready/done`, `get_timestamp`) and emits events for scheduling, execution, cancellation, and min-delay changes.

Token Module Updates

Fungible Tokens: Muxed Destinations and Allowance Hygiene

Transfers now accept `MuxedAddress` and emit `to_muxed_id` in events. Allowlist/blocklist overrides and macros have been updated accordingly. RWA token transfers delegate to the underlying muxed-aware path. Expired allowances now resolve to zero automatically instead of returning stale amounts.

Non-Fungible Metadata Guards

Enforces maximum lengths for name (40) and symbol (10) with new errors `NameMaxLenExceeded` and `SymbolMaxLenExceeded`, in addition to existing base-URI limits.

RWA Identity Registry Updates

Country-data metadata is bounded to 10 entries and 100 characters per value. Violations raise `MetadataTooManyEntries` / `MetadataStringTooLong`, and validation now runs on add/modify paths. The exported `validate_country_data` helper now supports reuse.

Trust Assumptions

The following assumptions underpin the safety of the newly introduced or modified components:

- **Soroban Platform**: Host cryptography, ledger timestamps, and keccak256 remain correct and tamper-resistant for fixed-point math, timelock hashing, and auth flows.
- **Contract Integrators**: Projects embedding fee abstraction or timelock must enforce their own authorization (who can set min_delay, allow fee tokens, sweep funds, schedule/execute/cancel), and simulate forwarded calls to avoid misuse.
- **Administrators**: Parties managing roles and smart-account policies are expected to respect new limits (`MAX_ROLES`, context rule caps) and resolve pending admin transfers before renouncing.
- **Token Issuers**: Fungible/NFT/RWA issuers correctly validate `MuxedAddress` recipients off-chain where relevant and provide compliant metadata within new bounds.
- **Off-chain Signers/Verifiers**: WebAuthn and policy signers properly manage keys and thresholds to match on-chain configurations, especially after duplicate-install protections.

High Severity

H-01 Off-By-One Rounding for Double Negatives

In `i128_fixed_point.rs`, `div_floor(-5, -2)` returns `1` even though the mathematical floor of `(-5)/(-2)` is `2`, so any call to `scaled_mul_div_floor` or its i256 counterparts underpays whenever a positive quotient arises only because both the operands were negative. For example, in `i256_fixed_point.rs`, `mul_div_floor(env, &I256::from_i32(env, -1), &I256::from_i32(env, 1), &I256::from_i32(env, -2))` yields `-1` instead of `floor(-1*1/-2)= floor(0.5)= 0`.

The rounding helpers determine the adjustment solely from the sign of the intermediate product `r`, assuming `r < 0` implies a negative quotient and `r > 0 && z < 0` implies the same. When both `r` and `z` are negative, this assumption fails because the true quotient is positive, yet the branch still subtracts one (or skips the ceiling bump), producing an off-by-one result across `div_floor`, `div_ceil`, and the i256 `mul_div_{floor,ceil}` and `checked_mul_div_{floor,ceil}` helpers.

Consider computing the quotient sign explicitly and only rounding down when the quotient is actually negative.

Update: *Resolved in pull request #521.*

Medium Severity

M-01 Fee Collection Can Be Decoupled From Authorization

`auth_user_and_invoke` authenticates `(fee_token, max_fee_amount, expiration_ledger, target_contract, target_fn, target_args)` and immediately spends the Soroban nonce but does not collect fees, leaving `collect_fee` to be invoked later with unchecked arguments. If these two functions are not bundled together in a contract implementation, a malicious relayer can submit a legitimate authorization for token

`A` and fee `10`, and then invoke `collect_fee` directly with token `B` and fee `1_000` provided the user has given valid approval to the fee-forwarder contract earlier, resulting in an unauthorized transfer. This is because the second helper never re-verifies the user input and this can be replayed until the all pre-approved tokens are exhausted.

The root cause is the separation between authorization and fee transfer. `collect_fee` trusts its caller and neither invokes `require_auth_for_args` nor validates its parameters against any stored authorization record. Since the earlier helper has already consumed the implicit nonce, it is impossible to reapply the same signature, and no explicit nonce or hash binding is persisted to bridge the two steps. The lack of linkage between authorization and transfer undermines both financial safety and the reliability of fee forwarding.

Consider moving the authorization into `collect_fee` (or a wrapper that invokes it immediately) and extending the signed payload with an explicit nonce. Alternatively, consider storing and validating a hash of the authorized parameters before executing `approve` and `transfer_from`.

Update: *Resolved in pull request #546.*

M-02 `checked_div` Overflows Legitimate Ratios

`Wad::checked_div` rejects large-but-valid ratios because it multiplies the numerator by `WAD_SCALE` before dividing. As a result, the intermediate `i128` product can overflow even when the final quotient fits. For example, `let a = Wad::from_integer(&env, 5_000); let b = Wad::from_integer(&env, 5_000); assert_eq!(a.checked_div(b), None);` returns `None` despite the expected result being exactly `1e18`, which should be representable. This lets untrusted inputs force otherwise safe conversions to fail.

The implementation performs `self.0.checked_mul(WAD_SCALE)` but does not fall back to the `i256` "phantom overflow" helpers that protect other math operations, such as in `checked_mul`, which routes through `checked_fixed_mul_floor`. This may result in denial of service for high-value trades and settlements. Honest actors attempting to divide large positions also suffer unnecessary failures, reducing system liveness and undermining confidence in the math library.

Consider implementing intermediate phantom overflow in `checked_div` to allow an equivalent extended-precision path as in `checked_mul`.

Update: *Resolved in pull request #539.*

M-03 Checked Fixed-Point Helpers Revert on Intermediate I256 Overflow

`SorobanFixedPoint`'s checked helpers in `checked_mul_div_floor` compute `r = x.mul(y)` before any division or error handling. So, operands such as `x = 2^155` and `y = 2^101` overflow the intermediate product even though `(x * y) / 2^101` would fit in 256 bits. A transaction that calls `checked_fixed_mul_floor` with such operands cannot receive `None` and instead encounters a host revert, contradicting the documented behavior of the checked API. This also applies to `checked_mul_div_ceil`.

This flaw lets untrusted inputs trigger deterministic denial of service for any contract logic that depends on `Option<I256>` to represent invalid fixed-point operations. Callers that expect to inspect a `None` result lose control flow to a hard revert, undermining reliability and potentially bricking batched transactions whenever large decimals propagate through these math helpers.

Consider replacing the direct `x.mul(y)` calls with a checked version for overflow and propagating the resulting `None`.

Update: *Acknowledged, will resolve. The team stated:*

> *When `soroban_sdk` can expose the `checked` versions of the `mul`, `div`, etc. we can just replace them, and the fix should be complete.*

Low Severity

L-01 Misleading `DivisionByZero` Error on Overflow

In `i128_fixed_point.rs`, calling `scaled_mul_div_floor(&i128::MIN, &env, &1, &-1)` will flow into `div_floor`, take the negative branch, and immediately evaluate `i128::MIN / -1`, which overflows. Instead of returning `SorobanFixedPointError::Overflow`, it returns the incorrect error SorobanFixedPointError::DivisionByZero. The subsequent unwrap in `scaled_mul_div_floor` collapses every `None` from `div_floor` into `SorobanFixedPointError::DivisionByZero`, so genuine overflow cases are surfaced with the misleading error.

Consider updating the unwrapping operation to distinguish an overflow from true zero divisors to provide accurate diagnostics.

Update: *Resolved in pull request #537.*

L-02 `Wad::checked_mul` and `Wad::pow` Rounding Behavior Disagrees With Documentation for Negative Results

The `Wad::checked_mul` method uses floor rounding (toward negative infinity) for negative products. However, its documentation claims that it truncates toward zero. This creates an inconsistency with the `Mul` trait implementation and may cause unexpected off-by-one errors in financial calculations. A similar issue exists in the `Wad::pow` method, which has the same comment while also using floor rounding.

Consider rounding in the correct direction to be consistent with the documentation when the result is negative.

Update: *Resolved in pull request #544.*

L-03 Unchecked Current Contract Address In `collect_fee`

`collect_fee` accepts `user` and `fee_recipient` as parameters and, with the allowlist disabled, permits any token. It validates fee bounds, then either unconditionally approves (Eager) or checks existing allowance and conditionally approves (Lazy). Finally, it performs `transfer_from` with `spender` set to the current contract. However, the function does not restrict `user` from being the current contract.

In the token implementation, `approve` only requires owner authorization and `transfer_from` only requires spender authorization. When `user` equals the current contract, both checks succeed for contract-initiated calls, enabling arbitrary callers of `collect_fee` to route contract-held funds to any `fee_recipient` when the integrating contract exposes `collect_fee` permissionlessly. While `collect_fee` is expected to be gated by `auth_user_and_invoke`, this is not guaranteed and the potential for misuse exists.

Consider adding an explicit precondition to reject `user == e.current_contract_address()` in `collect_fee` to prevent self-approval and self-transfer patterns that drain the contract's own holdings.

Update: *Resolved in pull request #542.*

Notes & Additional Information

N-01 Misleading Error Message in `get_country_data`

The `get_country_data` function is documented to only fail with `CountryDataNotFound` when the index is out of bounds. However, the implementation first obtains the profile via `get_identity_profile`, which panics with `IdentityNotFound` if the profile is absent. This error is not remapped by `get_country_data` prior to accessing the vector entry.

Consider aligning behavior and documentation by either remapping a missing profile to `CountryDataNotFound` inside `get_country_data`, or updating the function's documentation to include `IdentityNotFound`.

Update: *Resolved in pull request #541.*

N-02 Fingerprint Excludes Expiration Despite Documentation

The fingerprinting function is documented to include signers, policies, and expiration, as shown in the comments above `compute_fingerprint`. However, the implementation excludes `valid_until`, hashing only `context_type`, sorted signers, and sorted policies. Deduplication relies solely on this fingerprint via `validate_and_set_fingerprint` and is invoked on rule creation in `add_context_rule`.

Consider either updating the documentation or the fingerprint implementation to include the expiration to improve consistency.

Update: *Resolved in pull request #540.*

N-03 Incorrect Documentation Regarding Max Roles

The access-control documentation asserts that “one can create as many roles as they want”, as written in the module docs. The implementation, however, defines a `MAX_ROLES` of 256 and enforces this cap in `add_to_role_enumeration`.

Consider correcting the documentation such that it aligns with the implementation to avoid confusion.

Update: *Resolved in pull request #538.*

Conclusion

The OpenZeppelin Stellar Contracts library continues to evolve meaningfully, making big strides in safety rails and composability. The new fee abstraction and timelock primitives expand the library's governance and UX surface while keeping authorization consciously out-of-scope for integrators. Access-control hardening (role enumeration limits, safer admin renounce) and policy install guards reduce footguns, though they still rely on disciplined administrators. Math and token-layer upgrades (checked fixed-point, muxed transfers, stricter metadata bounds) improve correctness while guarding against failures at the edge cases.

Overall, the changes reinforce an already solid foundation, but their security depends on integrators carefully configuring aspects such as smart account policies, access-control delays, and fee approvals with equal care.

Appendix

Issue Classification

OpenZeppelin classifies smart contract vulnerabilities on a 5-level scale:

- Critical
- High
- Medium
- Low
- Note/Information

Critical Severity

This classification is applied when the issue's impact is catastrophic, threatening extensive damage to the client's reputation and/or causing severe financial loss to the client or users. The likelihood of exploitation can be high, warranting a swift response. Critical issues typically involve significant risks such as the permanent loss or locking of a large volume of users' sensitive assets or the failure of core system functionalities without viable mitigations. These issues demand immediate attention due to their potential to compromise system integrity or user trust significantly.

High Severity

These issues are characterized by the potential to substantially impact the client's reputation and/or result in considerable financial losses. The likelihood of exploitation is significant, warranting a swift response. Such issues might include temporary loss or locking of a significant number of users' sensitive assets or disruptions to critical system functionalities, albeit with potential, yet limited, mitigations available. The emphasis is on the significant but not always catastrophic effects on system operation or asset security, necessitating prompt and effective remediation.

Medium Severity

Issues classified as being of medium severity can lead to a noticeable negative impact on the client's reputation and/or moderate financial losses. Such issues, if left unattended, have a moderate likelihood of being exploited or may cause unwanted side effects in the system.

These issues are typically confined to a smaller subset of users' sensitive assets or might involve deviations from the specified system design that, while not directly financial in nature, compromise system integrity or user experience. The focus here is on issues that pose a real but contained risk, warranting timely attention to prevent escalation.

Low Severity

Low-severity issues are those that have a low impact on the client's operations and/or reputation. These issues may represent minor risks or inefficiencies to the client's specific business model. They are identified as areas for improvement that, while not urgent, could enhance the security and quality of the codebase if addressed.

Notes & Additional Information Severity

This category is reserved for issues that, despite having a minimal impact, are still important to resolve. Addressing these issues contributes to the overall security posture and code quality improvement but does not require immediate action. It reflects a commitment to maintaining high standards and continuous improvement, even in areas that do not pose immediate risks.