OpenZeppelin | security

Stellar Contracts Library 0.1.0-RC Audit

February 20, 2025

Table of Contents

Summary

Type	Library
Timeline	From 2025-02-03 To 2025-02-07
Languages	Rust (Soroban)

Total Issues	15 (14 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	1 (1 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	5 (4 resolved)
Notes & Additional Information	7 (7 resolved)
Client Reported Issues	1 (1 resolved)

Scope

We audited the OpenZeppelin/stellar-contracts repository at commit 01dbcb5.

In scope were the following files:

```
contracts
├── token/fungible/src
│   ├── extensions
│   │   ├── burnable
│   │   │   ├── mod.rs
│   │   │   └── storage.rs
│   │   ├── metadata
│   │   │   ├── mod.rs
│   │   │   └── storage.rs
│   │   ├── mintable
│   │   │   ├── mod.rs
│   │   │   └── storage.rs
│   │   └── mod.rs
│   ├── fungible.rs
│   ├── lib.rs
│   └── storage.rs
└── utils
    ├── pausable
    │   └── src
    │       ├── lib.rs
    │       ├── pausable.rs
    │       └── storage.rs
    └── pausable-macros
        └── src
            ├── helper.rs
            └── lib.rs
```

System Overview

The Soroban mainnet launched in February 2024, making it possible to deploy smart contracts to the Stellar blockchain. It marked a pivotal moment in the evolution of the Stellar ecosystem. One year later, the Stellar Development Foundation (SDF) joined forces with OpenZeppelin to simplify and accelerate the development process for projects building on Soroban.

The Stellar Contracts Library is a focused set of smart contract components designed to foster the development of robust, secure, and efficient decentralized applications. This audit examined the first release candidate of the library, which currently consists of two key modules: the fungible token and its extensions, and a pausable utility.

Fungible Token

Developed according to the SEP-0041 standard, the `openzeppelin_fungible_token` crate includes essential token extensions: burnable, metadata, and mintable, which are described in more detail below. The fungible token contract module implements utilities for managing fungible tokens within a Soroban contract. It provides essential storage capabilities for handling token balances, allowances, and the total supply.

Designed with flexibility in mind, this module divides its functionality into two levels. High-level functions encapsulate all necessary checks, verifications, authorizations, state changes, and event emissions, allowing users to perform standard token operations without delving into the underlying complexities. In contrast, low-level functions offer granular control for developers who wish to build custom workflows, exposing internal mechanisms that require manual handling of verifications and authorizations.

Burnable

The `FungibleBurnable` trait may be used to extend the `FungibleToken` trait to provide the capability to burn tokens. The `burnable` module contains three functions: `burn` and `burn_from`, which are responsible for removing tokens from a given account from circulation, and `emit_burn`, which is responsible for emitting a `burn` event.

Mintable

The `FungibleMintable` trait may be used to extend the `FungibleToken` trait to provide the capability of minting tokens. This trait is designed to be used in conjunction with the `FungibleToken` trait. The `mintable` module contains the `mint` function, which creates new tokens for a given account, and the `emit_mint` function, which emits the `mint` event.

Metadata

Unlike other extensions, `metadata` does not provide a separate trait because the corresponding functions are already available in the `FungibleToken` trait. It contains functions responsible for setting and returning the metadata related to the token, such as its symbol, name, and decimals.

Pausable Utility

This module offers robust mechanisms for managing contract states during emergencies, ensuring that projects can respond effectively to critical situations. It includes the `Pausable` trait required to be implemented by any contract that handles pausing and unpausing. In addition, two helper attribute macros, `when_paused` and `when_not_paused`, are provided to act as execution gates, checking whether the contract is paused before execution.

Security Model and Trust Assumptions

The Stellar Contracts Library relies fundamentally on the security of the Soroban SDK and the supporting crates that facilitate secure macro development. We assume that these out-of-scope dependencies are both safe and actively maintained. Moreover, the OpenZeppelin contracts library has been designed for maximum flexibility. Thus, users are responsible for integrating its functions according to the recommended guidelines and must exercise caution when customizing implementations, as such modifications could introduce vulnerabilities.

High Severity

H-01 Attribute Macros Omit Subsequent Attributes

The `when_not_paused` and `when_paused` attribute macros insert a pause check at the beginning of a function to determine whether the contract is paused. Within their output blocks [1] [2], they process the consumed function, preserving its visibility and signature, and then apply the appropriate pause check before returning the function's body.

However, the generated output does not preserve subsequent attributes. As a result, when a function is annotated with `when_not_paused` or `when_paused` followed by additional attributes, those attributes are omitted. This can lead to unintended behavior, for example, in cases where `when_paused` is followed by `only_owner` to restrict access to the contract owner after pausing as a safety measure. Since `only_owner` appears after `when_paused`, it gets ignored, potentially allowing unauthorized access to a function intended only for the owner.

To prevent this issue from happening, consider modifying the macro logic to retain and correctly apply subsequent attributes when generating function output.

Update: *Resolved in pull request #29 at commit ff978d3.*

Medium Severity

M-01 Approval Periods Are Implicitly Restricted

The `approve` function defined in the `openzeppelin_fungible_token` crate authorizes the spender to spend the `amount` of tokens from the owner's account. The approval is valid until the ledger number specified in the `live_until_ledger` argument. The difference between this value and the current ledger number is then used in order to extend the approval's entry TTL. However, there is a maximum period that each entry may be extended by at a time, and if the difference between the `live_until_ledger` argument and the current ledger number is higher than that, the code will panic and the approval will not be made. As a

result, it is not possible to approve tokens for a time frame longer than the maximum TTL value.

Consider allowing accounts to specify approvals that are valid for longer than the maximum TTL value by limiting the TTL extension period to the maximum possible value. It should be clearly documented that it is the responsibility of the approving account or the approval recipient to extend the TTL of the storage entry containing the approval data so that it does not expire until `live_until_ledger`. Alternatively, consider using `permanent` storage for storing allowances.

Update: *Resolved in pull request #57 at commit 269a679 by documenting the design choice of restricting the approval periods to the maximum TTL value. It is worth noting that the approval logic currently slightly differs from the one in the Stellar Asset Contract (SAC) as approval storage entry TTL is extended by the entire approval period, not the approval period increased by 1 as in the SAC. However, this logic in the SAC may be changed in the future, so that the TTLs of approval storage entries will not be increased by 1 anymore as indicated by this issue.*

Low Severity

L-01 `Env` Type Check Can Be Bypassed

In the `openzeppelin_pausable_macros` crate, the `check_is_env` function verifies whether the first argument of a function is of type `soroban_sdk::Env`. However, the check currently relies only on the type name `"Env"`, making it susceptible to bypasses. For instance, if `Env` is aliased (e.g, `use soroban_sdk::{Env as Environment}`), the check will fail since the type name `"Environment"` does not match the name `"Env"`. In addition, issues arise when using an `Env` type from another crate, for example, `use external_crate::Env`, or aliasing an unrelated type as `Env`, such as `use external_crate::{RandomType as Env}`.

To ensure accuracy, consider resolving the full path of the argument's type to explicitly verify that it matches `soroban_sdk::Env`.

Update: *Acknowledged, not resolved. The OpenZeppelin team stated:*

> *Procedural Macros in Rust cannot enforce the type safety. In this case, the macro cannot enforce `e: &Env` belongs to `soroban_sdk::Env`, because macros can only access the scope they are annotating, and does not have access to the other parts of*

> *the code, hence, the macro cannot look at the imports, nor analyze the full type during expansion. The responsibility of not shadowing or overwriting the `Env` variable belongs to the developer, not to the macro.*

L-02 Misleading Documentation

Throughout the codebase, multiple instances of misleading documentation were identified:

- In the `openzeppelin_fungible_token` crate's `lib.rs` file, the documentation states that `Mintable` allows authorized entities to mint. However, the documentation for the `mint` function clearly states that it lacks authorization control and that authorization is the responsibility of the implementer.
- The `approve` function is missing the error documentation of `FungibleTokenError::LessThanZero`.
- In the `openzeppelin_fungible_token::extensions::mintable` crate, the comment in line 31 of `storage.rs` assumes that `require_auth` implements the `Try` trait by using the `?` for error handling, which is incorrect.
- The `burn` function is missing error documentation for `FungibleTokenError::LessThanOrEqualToZero`. The same is true for the `burn_from` function.
- The documentation of `burn` and `burn_from` functions states that they destroy `amount` of tokens from `account`, whereas in reality, they do not have the `account` parameter, and they destroy `amount` of tokens from the `from` account.
- The documentation of the `FungibleBurnable` and `FungibleMintable` traits refers to them as `Burnable` and `Mintable`, respectively, not including the "Fungible" prefix.

Consider addressing the above instances of misleading documentation to improve the clarity and maintainability of the codebase.

Update: *Resolved in pull request #50 at commit 1388d4d.*

L-03 Transfers of 0 Amount Not Possible

The `update` helper function defined in the `openzeppelin_fungible_token` crate is responsible for updating token balances and the total supply of tokens based on the provided arguments. However, this function panics when the specified amount of tokens equals 0. As a result, transfers of 0 tokens are not possible. While this is not required by the SEP-41 standard,

the ERC-20 standard, widely used on EVM-compatible blockchains, requires such transfers to be possible.

Consider allowing 0-token transfers in order to provide contracts utilizing these tokens with more flexibility.

Update: *Resolved in pull request #48 at commit 129dfe1.*

L-04 Instance TTL Not Updated on All Operations

Whenever operations modifying the data stored in the `instance` storage of the token contract such as minting or burning tokens are performed, the entire `instance` TTL is increased. However, other operations performed on the token contract, such as querying balance or transferring tokens between accounts, do not increase the `instance` TTL. As a result, it is possible that the token contract is used, but its data eventually becomes archived. As a result, it will have to be manually restored, which incurs an additional cost.

Consider increasing the contract's `instance` storage TTL on all operations querying or modifying the state of the contract. Alternatively, consider documenting the current design choice so that it is clear for the implementers.

Update: *Resolved in pull request #62 at commit 4500bfb.*

L-05 Insufficient Validation

The `spend_allowance` helper function implemented in the `openzeppelin_fungible_token` crate deducts the given amount of tokens from the `owner`'s allowance for the `spender`. However, it does not contain a check against the specified `amount` being negative. Hence, it will not panic in such a case.

As long as this function is used along with the `update` function, which performs such a check, it will not cause any problems. But if the `spend_allowance` function is used in a different context, possibly with a different implementation of the `update` function, it could result in surprising consequences where an incorrect amount of tokens is accounted for.

Consider making the `spend_allowance` function more self-contained by panicking inside the function in case the specified `amount` is negative.

Update: *Resolved in pull request #49 at commit 12f81e6.*

Notes & Additional Information

N-01 Duplicated Code

The `when_not_paused` and `when_paused` macros contain nearly identical code, leading to redundancy.

To enhance maintainability, readability, and conciseness, consider refactoring these macros and reducing code duplication.

Update: *Resolved in pull request #29 at commit 544354d.*

N-02 Potentially Unused Variable

The `default` variable in the `allowance_data` function is returned when there is no allowance or when `live_until_ledger` is less than the current ledger number. However, when there is an allowance value to return, the `default` variable remains unused.

Consider removing the unnecessary variable declaration to improve code clarity and efficiency.

Update: *Resolved in pull request #56 at commit 615e13d.*

N-03 Unnecessary Check

In the `openzeppelin_fungible_token` crate, the `allowance` function should return 0 when the `live_until_ledger` value is less than the current ledger number. The code in line 105 checks if `live_until_ledger` value is less than the current ledger number and if the allowance is bigger than 0. The latter check is unnecessary and can be removed since it has no effect on the logic and the outcome of the function.

Consider removing any unnecessary checks to improve code clarity.

Update: *Resolved in pull request #55 at commit 778a6e7.*

N-04 Unclear Revert Reason

In the `openzeppelin_fungible_token::extensions::metadata` crate, the `get_metadata` function is used to return the token metadata such as decimals, name, and symbol. The function attempts to get the metadata from the storage and will panic if no value is found using `unwrap_optimized`. Usually, it is preferred to revert using `unwrap_or` or similar error-handling functions to return a default value. However, in this case, returning a fixed default can be misleading in some cases and may alter the token's behavior. Furthermore, it is expected that the metadata is always set at deployment in the constructor using the `set_metadata` function.

To prevent returning a default value but reverting with a more descriptive error, consider unwrapping the value using `expect`, thus reverting with a descriptive message to ease debugging in case of an error.

Update: *Resolved in pull request #54 at commit 5737528.*

N-05 Typographical Errors

Throughout the codebase, multiple instances of typographical errors were identified:

- In line 11 of `token/fungible/src/fungible.rs`, "have" should be "has".
- In line 61 of `token/fungible/src/extensions/burnable/mod.rs`, "A" at the end of the line could be removed.
- In line 379 of `token/fungible/src/storage.rs`, "amoount" should be "amount".
- In line 37 of `token/fungible/src/extensions/burnable/storage.rs`, there is no space between `amount` and "is".

Consider correcting all instances of typographical errors in order to improve the clarity and readability of the codebase.

Update: *Resolved in pull request #52 at commit 51e84b0.*

N-06 Rephrasing Suggestions

The documentation of the pausable module contains a statement that could be made more formal. Furthermore, "due to" in the following line could be changed to "due to the following reasons" in order to enhance readability.

In order to enhance the readability of the codebase, consider rephrasing the comments mentioned above.

Update: *Resolved in pull request #53 at commit 23a21e6.*

N-07 Code Inconsistency

Throughout the codebase, the `panic_with_error` macro is generally used without a semicolon at the end of the line. However, three instances in `storage.rs` within the `openzeppelin_fungible_token` crate [1] [2] [3] include a semicolon. While this does not pose a security risk, it introduces code inconsistency and may cause confusion.

To enhance consistency and readability, consider adopting a uniform formatting style across the codebase.

Update: *Resolved in pull request #73 at commit 29696bc.*

Client Reported

CR-01 Lack of Necessary Derives for the Contract Errors

The `FungibleTokenError` and `PausableError` error enums defined in the `openzeppelin_fungible_token` and `openzeppelin_pausable` crates do not follow the requirements from the documentation. Particularly, the error definitions do not use the `#[repr(u32)]` and `#[derive(Copy)]` attributes.

To follow best practices, consider implementing the derive macros suggested by the docs when defining contract errors with the `#[contracterror]` attribute.

Update: *Resolved in pull request #31 at commit 7342590.*

Conclusion

The first release candidate of the Stellar Contracts Library is an initial effort to standardize common contracts, particularly the fungible token and pausable utility, and to streamline development and encourage adoption on Soroban. Overall, the codebase is concise, adheres to good coding practices, and includes extensive documentation and testing.

During the audit, one high-severity issue was identified. The Soroban Contracts team was highly responsive, actively engaging in discussions about design decisions and providing valuable insights into the library's development. We appreciate the SDF's dedication to creating a more accessible development environment that will ultimately benefit the entire ecosystem.