OpenZeppelin | security

Contracts for Stylus Audit

ARBITRUM FOUNDATION

OpenZeppelin

October 17, 2024

Table of Contents

Table of Contents 2

Summary 3

Scope 4

System Overview 6

Stylus and Rust SDK 6

OpenZeppelin Contracts for Stylus 6

Main Modules 6

Limitations of the SDK 7

Trust Assumptions 8

Medium Severity 9

M-01 ERC-721 Does Not Support ERC-165 9

Low Severity 9

L-01 Incomplete Implementation for checkpoints.rs 9

L-02 Inconsistent Implementations in ERC-721 Extension 10

L-03 Missing Checks in ERC-20 Implementation 10

L-04 Public Exposure of pause and unpause Functions 11

L-05 Inconsistent Function Visibility in Burnable Extension 11

L-06 Inconsistent Implementation Styles 12

Notes & Additional Information 12

N-01 Unaddressed TODOs and FIXMEs 12

N-02 Inconsistent Error Types 13

N-03 Unnecessary Swap Operation 13

N-04 Misleading Reference to Modifiers in Pausable Module 13

N-05 Unused Error Message 14

Conclusion 15

Summary

Type	Smart Contract Library
Timeline	From 2024-08-26 To 2024-10-01
Languages	Rust

Total Issues	12 (9 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	6 (4 resolved)
Notes & Additional Information	5 (4 resolved)

Scope

We audited the OpenZeppelin/rust-contracts-stylus repository at commit 42c859f.

In scope were the following files:

```
contracts/src
├── access
│   ├── control.rs
│   ├── mod.rs
│   └── ownable.rs
├── lib.rs
├── token
│   ├── erc20
│   │   ├── extensions
│   │   │   ├── burnable.rs
│   │   │   ├── capped.rs
│   │   │   ├── metadata.rs
│   │   │   ├── mod.rs
│   │   │   └── permit.rs
│   │   └── mod.rs
│   ├── erc721
│   │   ├── extensions
│   │   │   ├── burnable.rs
│   │   │   ├── consecutive.rs
│   │   │   ├── enumerable.rs
│   │   │   ├── metadata.rs
│   │   │   ├── mod.rs
│   │   │   └── uri_storage.rs
│   │   └── mod.rs
│   └── mod.rs
└── utils
    ├── cryptography
    │   ├── ecdsa.rs
    │   ├── eip712.rs
    │   └── mod.rs
    ├── math
    │   ├── alloy.rs
    │   ├── mod.rs
    │   └── storage.rs
    ├── metadata.rs
    ├── mod.rs
    ├── nonces.rs
    ├── pausable.rs
    └── structs
        ├── bitmap.rs
        ├── checkpoints.rs
        └── mod.rs
lib/crypto
```

```
└── src
    ├── hash.rs
    ├── keccak.rs
    ├── lib.rs
    └── merkle.rs
```

System Overview

Stylus and Rust SDK

Stylus is an upgrade to Arbitrum Nitro that introduces a second, coequal virtual machine to the EVM. This allows EVM contracts to function as they do on Ethereum within the first VM, while the second VM executes Stylus contracts that use WebAssembly (WASM) instead of EVM bytecode. The WASM code is written in Rust using the Arbitrum Stylus Rust SDK, which is built on top of Alloy, a collection of crates designed to empower the Rust Ethereum ecosystem. By leveraging the same Rust primitives for Ethereum types, the Stylus SDK remains compatible with existing Rust libraries.

OpenZeppelin Contracts for Stylus

The Contracts for Stylus library, the subject of the present audit, has been designed to provide a foundation for smart contract development on Arbitrum Stylus, closely following the design and practices of the `openzeppelin-contracts` library for Solidity. It builds upon the Stylus SDK which is written in Rust. While the initial implementation of the Contracts for Stylus library is intended to be a port of the Solidity library and not stray too far from the heavily audited business logic therein, current limitations of the Stylus SDK require some workarounds and custom features in order to provide the intended functionality. This is primarily due to the fact that the SDK, and Arbitrum Stylus in general, is still in early stages of development, and the Contracts for Stylus library team continues to work closely with developers from Arbitrum Stylus in order to advance both codebases in parallel.

Main Modules

The following are the main modules of the Contracts for Stylus library.

Access

This module includes functionality for ownership and role-based access control. The `Ownable` trait allows a contract to designate a single owner with privileged actions, while `AccessControl` enables the creation of multiple roles with specific permissions (e.g., minting

or burning tokens). This allows for flexible and granular access management, enhancing security and composability across contracts.

Token

Provides the implementation for the ERC-20 and ERC-721 token standards along with various extensions. Extensions for ERC-20 such as Burnable, Capped, Pausable, and Permit add advanced functionality like destroying tokens, enforcing supply limits, pausing transfers, or allowing gasless approvals (EIP-2612). Extensions for ERC-721 include Burnable, Enumerable, and Consecutive. The Enumerable extension allows for enumeration of all the token IDs in the contract as well as all the token IDs owned by each account. On the other hand, the Consecutive extension is useful for efficiently minting multiple tokens in a single transaction.

Crypto

The Crypto module provides cryptographic utilities commonly used in blockchain environments, including functionality for verifying Merkle proofs. It allows developers to verify whether an element is part of a Merkle tree using the `verify` and `verify_multi_proof` functions. These functions leverage the Keccak256 hashing algorithm by default but can be customized to use generic hashing algorithms through `verify_with_builder`. This makes the module versatile for use in whitelisting and other proof-based applications.

Limitations of the SDK

The Stylus SDK presents several challenges that complicate replicating OpenZeppelin's Solidity libraries. These include:

Lack of Constructors

OpenZeppelin developed Koba to allow Stylus contracts to be deployed with input, mimicking constructor behavior. However, development is still in early stages and not all functionality is currently available, such as checking the `code.length` of the current contract to determine if it is at the deployment stage.

Lack of Modifiers

Since there is no modifier support, the only solution for contracts such as `Ownable` is to define two functions, one to be called at the beginning of execution and one to be called at the end, which is an error-prone pattern when compared to Solidity.

Limited Inheritance

Since inheritance is not available, certain workarounds are required in the library to override internal functions. For example, `consecutive.rs` has to re-implement many functions of the base `ERC721` in order to add additional logic, which leads to code duplication.

Trust Assumptions

It is assumed that the Stylus SDK and other third-party dependencies integrated with the Contracts for Stylus library are trusted to be secure and behave in an expected manner, and to be free from malicious code such as backdoors or other vulnerabilities. In addition, several tools are closely integrated within the Contracts for Stylus library that were developed by OpenZeppelin to provide additional custom functionality. Koba was developed to provide constructor support, while Motsu and the E2E Testing Crate were developed to add unit and integration tests, respectively, to the Stylus development workflow. These crates are out-of-scope for the current audit and have not yet undergone an audit at this time.

Medium Severity

M-01 ERC-721 Does Not Support ERC-165

The ERC721 implementation does not support ERC-165, which is required by the interface, and popular marketplaces such as OpenSea and Rarible call `supportsInterface` on any NFTs they interact with. If an ERC-721 implementation does not implement ERC-165, other contracts, dApps, or marketplaces will not be able to easily verify whether the contract supports the ERC-721 interface. This means that platforms or third parties that rely on ERC-165 to check for compliance will not recognize the contract as an ERC-721 token. As a result, the NFT contract may face limited interoperability and integration issues with external systems that expect proper ERC-165 implementation.

Consider adding support for ERC-165 to ensure compatibility with marketplaces and third-party systems.

Update: *Resolved at commit 4106d03, ERC-165 support was added.*

Low Severity

L-01 Incomplete Implementation for `checkpoints.rs`

The `checkpoints.rs` file in the Stylus library only includes an implementation for the `uint160` type. It lacks implementations for the `uint208` and `uint224` types, which are part of the Solidity library. A TODO comment on line 17 indicates that this is intended to be addressed in the future.

Consider adding implementations for the missing types to ensure consistency with the Solidity library.

Update: *Resolved at commit 3b560b3.*

L-02 Inconsistent Implementations in ERC-721 Extension

The Stylus library ERC-721 extensions are not consistent with the respective functionality of the Solidity library.

These inconsistencies were identified:

- In the uri_storage extension, the `token_uri()` function directly returns the value from storage but omits key checks:
  - It does not verify whether the token exists, which is handled in the Solidity implementation.
  - It does not handle base URI concatenation like the Solidity version. Instead, it directly retrieves and returns the stored token URI without constructing the full URI.
- There is a mismatch in the implementation of `ERC721Metadata`:
  - The Rust implementation exposes the `base_uri` function. In the Solidity version, URIs are only accessible by token ID when the token exists. Not exposing `baseURI` is generally advisable, as revealing it prematurely can leak sensitive NFT data before minting. This can enable advanced users to time their minting to acquire more desirable NFTs.
  - The Rust version lacks the `tokenURI` function by default, which may cause integration issues.

Consider aligning these implementations with the Solidity library to ensure consistency and mitigate potential risks.

Update: *Resolved at commit 47d2d9c.*

L-03 Missing Checks in ERC-20 Implementation

Some validation checks are missing in the current ERC-20 implementation when compared to the Solidity library:

- In `mod.rs`, the `_approve()` function checks the spender's address but does not verify that the owner's address is not zero, unlike in the Solidity library.

- The `_spend_allowance()` function directly sets the allowance in storage without calling the internal `_approve()` function, as is done in the Solidity version. This bypasses the address validation checks (owner and spender) that `_approve()` provides.
- The `transfer_from()` function currently calls `_spend_allowance()` before `_transfer()`, where the latter includes non-zero address checks for both the sender and recipient. However, these checks should be performed prior to updating the allowance, as is done in the Solidity implementation, to prevent unnecessary storage writes in the event of a revert.

Consider adding the aforementioned missing checks to improve the security of the library.

Update: *Resolved at commit 59dafb4.*

L-04 Public Exposure of `pause` and `unpause` Functions

The `pausable.rs` contract exposes the `pause` and `unpause` functions as part of a `#[public]` implementation block. This design automatically makes these functions available in any inheriting contract's public ABI, without any access control restrictions. In the Solidity version of the `Pausable` contract, the corresponding functions are marked as `internal`. This forces developers to explicitly declare `external` or `public` functions that interact with `pause` and `unpause`, and ensures that proper access control measures are applied.

Consider placing the `pause` and `unpause` functions in a non-public implementation block, preventing their exposure via the ABI by default. This will enforce explicit access control, reducing the risk of unintended function access.

Update: *Resolved at commit af826da.*

L-05 Inconsistent Function Visibility in Burnable Extension

The ERC-20 burnable extension implementation does not mark the public `burn` and `burn_from` functions as `external`, and developers will have to implement these by themselves. A similar issue is also found in the burnable extension for ERC-721. Although the documentation clearly states this, these design patterns are inconsistent with other extensions of the library.

In order to avoid confusion and make the library easy to use, consider using a consistent design pattern for the different extensions.

Update: *Acknowledged, not fixed due to the Stylus SDK restriction of only allowing one* `#[public]` *attribute declaration per crate.*

L-06 Inconsistent Implementation Styles

Throughout the codebase, multiple inconsistencies were identified in the implementation style, mostly related to limitations around inheritance. For example, the ERC721Consecutive and ERC20Permit contracts include an `erc721` or `erc20` element in a struct, respectively, and functions are called on that element when possible, but this forces the library to re-implement many functions from the base contract leading to code duplication. On another hand, the ERC20Capped seems somewhat incomplete compared to the Solidity library and would require the user to implement the capping logic. This mixed design style could be confusing and error-prone for developers using the library.

Consider adopting a consistent style for the codebase implementation or heavily document the different reasons for the decisions to help guide users adopt the best developer practices when using the library.

Update: *Acknowledged, not fixed due to Rust-based contract designs for Stylus not being compatible with Solidity-style inheritance patterns.*

Notes & Additional Information

N-01 Unaddressed TODOs and FIXMEs

Throughout the codebase, multiple instances of TODO and FIXME comments were identified:

- line59 of ERC-20 extension `metadata.rs`
- line73 of ERC-20 extension `metadata.rs`
- line11 of ERC-721 extension `enumerable.rs`
- line71 of ERC-721 extension `enumerable.rs`
- line486 of ERC-721 extension `mod.rs`

- line665 of ERC-721 extension `mod.rs`

During development, having well-described TODO or FIXME comments will make the process of tracking and resolving them easier. Without this information, these comments might age and important information for the security of the system might be forgotten by the time it is released to production. As such, consider removing all instances of TODO or FIXME comments and instead tracking them in the issues backlog. Alternatively, consider linking each inline TODO or FIXME to a corresponding backlog issue.

Update: *Resolved at commit 997fcc0.*

N-02 Inconsistent Error Types

Throughout the codebase, there are some extensions that use associated error types to allow users the flexibility to assign a generic `Vec` as an error type and encode any error that is needed (or not return an error at all), while others do not use this error style. For example, in the IERC20 trait of the ERC20 implementation.

Consider using a consistent error style to avoid confusion and improve code clarity.

Update: *Acknowledged, will standardize errors in a future release.*

N-03 Unnecessary Swap Operation

The `commutative_hash_pair` function in `hash.rs` performs a comparison of `a` and `b` values, swapping them if `a > b`. However, it would be more efficient and readable to simply reverse the order of arguments in the call to `hash_pair` if the comparison is true.

To improve code readability, consider modifying the `commutative_hash_pair` function to implement the aforementioned optimization.

Update: *Resolved at commit 7c84e68.*

N-04 Misleading Reference to Modifiers in Pausable Module

The Pausable module makes reference to modifiers before the `when_not_paused` and `when_paused` functions, but true modifiers are not supported by the Stylus SDK. Consider

changing the wording of these comments and any others in the codebase to ensure that there are references to features that are unique to Solidity.

Update: *Resolved at commit [c056473](#).*

N-05 Unused Error Message

The ECDSA library does not utilize the `ECDSAInvalidSignatureLength` error. According to the Solidity library, the concatenation of `v`, `r`, and `s` must have a valid signature length of 65 bytes, otherwise, an error is returned. However, this validation is not necessary in the Stylus contract implementation since the parameters are provided as fixed-length types. Consider removing this error definition to avoid confusion. In addition, consider correcting the misplaced comment referencing this error in line 189.

Update: *Resolved at commit [6eeb15c](#).*

Conclusion

The Contracts for Stylus library marks a significant step toward the maturation and adoption of Arbitrum Stylus as a smart contract language. For developers familiar with Solidity and the OpenZeppelin libraries but less experienced with Rust, this library provides an accessible onboarding path to Rust-based contract development while maintaining a secure level of abstraction over the Stylus SDK. During the security audit, several issues were identified, and we provided recommendations primarily focused on missing features and inefficient implementation patterns caused by SDK limitations. Since the library mirrors much of the Solidity implementation, these suggestions are in line with the evolving nature of both the library and the SDK.

Throughout the audit, we closely collaborated with the Contracts For Stylus team and anticipate significant developments as both the library and the SDK mature. To ensure ongoing security and functionality, we strongly recommend continuous audits as major updates occur. We are excited to see the project evolve, expand its scope, and contribute to the growing Arbitrum Stylus ecosystem.