OpenZeppelin | security

Stylus Contracts Library v0.2.0 Audit

June 17, 2025

Table of Contents

Summary

Type	
Timeline	From 2025-04-22 to 2025-05-14 (crypto library) and 2025-05-26 to 2025-06-06 (contracts library)

Languages

Total Issues	22 (20 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	1 (1 resolved)
Medium Severity Issues	2 (1 resolved)
Low Severity Issues	9 (9 resolved)
Notes & Additional Information	10 (9 resolved)
Client Reported Issues	0 (0 resolved)

Scope

We audited the OpenZeppelin/rust-contracts-stylus repository in two parts.

The first part was the crypto library. We audited the library at the a67ab70 commit from 2025-04-22 to 2025-05-09. Then, we audited the 2350766 commit, which introduced the `pedersen` directory below, from 2025-05-12 to 2025-05-14.

```
lib/crypto/src
├── arithmetic
│   ├── limb.rs
│   ├── mod.rs
│   └── uint.rs
├── bits.rs
├── const_helpers.rs
├── curve
│   ├── helpers.rs
│   ├── mod.rs
│   └── sw
│       ├── affine.rs
│       ├── mod.rs
│       └── projective.rs
├── field
│   ├── fp.rs
│   ├── group.rs
│   ├── instance.rs
│   ├── mod.rs
│   └── prime.rs
├── hash.rs
├── keccak.rs
├── lib.rs
├── merkle.rs
├── pedersen
│   ├── instance
│   │   ├── mod.rs
│   │   └── starknet.rs
│   ├── mod.rs
│   └── params.rs
├── poseidon2
│   ├── instance
│   │   ├── babybear.rs
│   │   ├── bls12.rs
│   │   ├── bn256.rs
│   │   ├── goldilocks.rs
│   │   ├── mod.rs
│   │   ├── pallas.rs
│   │   └── vesta.rs
│   ├── mod.rs
│   └── params.rs
└── test_helpers.rs
```

The second part was the contracts library, which we audited at the 5ed33dd commit. The following files were in scope:

```
contracts/src
├── access
│   ├── control.rs
│   ├── mod.rs
│   ├── ownable.rs
│   └── ownable_two_step.rs
├── finance
│   ├── mod.rs
│   └── vesting_wallet.rs
├── lib.rs
├── token
│   ├── common
│   │   ├── erc2981.rs
│   │   └── mod.rs
│   ├── erc1155
│   │   ├── extensions
│   │   │   ├── burnable.rs
│   │   │   ├── metadata_uri.rs
│   │   │   ├── mod.rs
│   │   │   ├── supply.rs
│   │   │   └── uri_storage.rs
│   │   ├── mod.rs
│   │   └── receiver.rs
│   ├── erc20
│   │   ├── extensions
│   │   │   ├── burnable.rs
│   │   │   ├── capped.rs
│   │   │   ├── erc4626.rs
│   │   │   ├── flash_mint.rs
│   │   │   ├── metadata.rs
│   │   │   ├── mod.rs
│   │   │   ├── permit.rs
│   │   │   └── wrapper.rs
│   │   ├── interface.rs
│   │   ├── mod.rs
│   │   └── utils
│   │       ├── mod.rs
│   │       └── safe_erc20.rs
│   ├── erc721
│   │   ├── extensions
│   │   │   ├── burnable.rs
│   │   │   ├── consecutive.rs
│   │   │   ├── enumerable.rs
│   │   │   ├── metadata.rs
│   │   │   ├── mod.rs
│   │   │   ├── uri_storage.rs
│   │   │   └── wrapper.rs
│   │   ├── interface.rs
│   │   ├── mod.rs
│   │   └── receiver.rs
│   └── mod.rs
└── utils
    ├── cryptography
    │   ├── ecdsa.rs
    │   ├── eip712.rs
    │   └── mod.rs
    ├── introspection
    │   ├── erc165.rs
    │   └── mod.rs
    ├── math
    │   ├── alloy.rs
    │   ├── mod.rs
    │   └── storage
    │       ├── checked.rs
    │       ├── mod.rs
    │       └── unchecked.rs
    ├── metadata.rs
    ├── mod.rs
    ├── nonces.rs
    ├── pausable.rs
    └── structs
        ├── bitmap.rs
        ├── checkpoints
        │   ├── generic_size.rs
        │   └── mod.rs
        └── mod.rs


contracts-proc/src
├── interface_id.rs
└── lib.rs
```

System Overview

Crypto Library

The Crypto Library consists of six major areas: the arithmetic folder, the curve folder, the field folder, the poseidon2 folder, the pedersen folder, and other modules in the root folder such as `hash.rs`, `keccark.rs`, `merkle.rs`, `bits.rs`, etc.

The `hash` module implements a generic hash function that supports larger and more secure outputs than the hasher in the core library, which only supports `u64` outputs. The `merkle` module offers functions to verify Merkle proofs.

The arithmetic module supports big integer operations by providing safe, efficient, and portable arithmetic on the underlying 64-bit chunks ("limbs") that make up large numbers.

The curve module defines the core traits and helpers needed to implement and use elliptic curves, supporting both affine and projective representations, efficient group operations, and batch field inversions. It serves as the foundation for higher-level cryptographic operations on elliptic curves.

The field module lays the foundation for finite field arithmetic, providing traits, implementations, and utilities for safe, efficient, and generic field operations—essential for cryptographic and algebraic computations.

The Poseidon2 module provides a fast and flexible implementation of the Poseidon2 hash function. The Pedersen module provides a fast implementation of the Pedersen hash function compatible with Starknet. They are suitable for modern cryptographic applications, especially in zero-knowledge and blockchain contexts.

Contract Library

The Stylus Smart Contracts Library has been extensively updated to align with the latest Stylus SDK conventions and to provide a broader suite of token utilities.

Constructors are now explicitly defined with the `#[constructor]` attribute and deployed atomically, ensuring that contract initialization cannot be skipped or manipulated. Error handling has been standardized using associated types, and all events now derive `Debug` for easier on-chain log inspection and off-chain debugging. The inheritance pattern has shifted toward the SDK's trait-based approach using the `#[implements(...)]` macro for trait-based inheritance.

On the feature side, the library introduces comprehensive support for multiple token standards:

- ERC-1155 now includes burnable extensions, metadata URI management, and supply controls.
- ERC-4626 vault functionality allows ERC-20 tokens to be wrapped into yield-bearing vaults.
- Flash-minting capabilities for ERC-20.
- A `SafeERC20` wrapper.
- A vesting wallet implementation.
- Several additional utilities.

The contract implementations closely follow the OpenZeppelin contracts library for Solidity due to its robust design, maximizing compatibility and consistency. ERC-165 is fully implemented across all contracts, so interface detection is reliable and consistent via `supports_interface`.

A new procedural macro was also introduced to generate an `interface_id` function for traits and handle custom selectors within those traits. Together, these enhancements improve both the expressiveness and safety of the library while preserving a clear, standardized design.

Security Model and Trust Assumptions

- The crypto library's Poseidon2 implementation matches the test vectors of the reference implementation.
- It is assumed that the Stylus SDK and all third-party dependencies used by the Contracts for Stylus library are secure, free from malicious code (such as backdoors), and behave as documented.
- Motsu, OpenZeppelin's internal testing utility for Stylus contracts, functions as intended, contains no critical bugs, and provides reliable coverage for the library's unit tests.

High Severity

H-01 Limb Shift Overflow in `shr_assign` and `shl_assign`

The `shr_assign` and `shl_assign` functions implement the shift-and-assign operators (`<<=` and `>>=`) for the `Uint<N>` type, which represents a multi-limb unsigned integer with `N` limbs. However, a shift overflow occurs [1] [2] when the `limb_shift` value is zero. In debug mode, this results in a runtime panic with the error message `attempt to shift left with overflow`. In release mode, the shift value is masked, producing incorrect results. Consider revising the code to address this issue before the overflow occurs.

***Update:** Resolved in pull #658 at commit c38b883. The team uses `checked_shl` to mitigate overflow situations.*

Medium Severity

M-01 Potential Underflow in `ct_rem` Function

The `ct_rem` function computes the remainder of a division operation. However, the calculation of `index` may result in an underflow if `self.ct_num_bits()` returns zero. This issue arises when the dividend (`self`) is zero, as `ct_num_bits` returns zero for a zero value. Subtracting one from zero in an unsigned integer (`usize`) causes an underflow, wrapping around to `usize::MAX` in release mode and leading to incorrect results for this function. It is advisable to return zero directly when the dividend is zero.

***Update:** Resolved in pull #667 at commit 36ab98e.*

M-02 Customer Selectors in Trait With `interface_id` Attribute Are Not Checked in Implementation

The `#[interface_id]` attribute is implemented as a procedural macro that can be added to a trait. This macro re-generates the trait source code with the addition of an `interface_id` function to produce its ERC-165 identifier.

The macro consumes any `#[selector]` attributes within the trait, allowing them to exist in traits. Within the Stylus SDK, the `#[selector]` attribute may only exist within `#[public]` blocks and does not provide logic for the macro within traits. However, there is no mechanism in place to enforce that trait implementations match the interface of traits with custom selectors.

As a consequence, developers must remember to add any custom selectors in trait implementations to correctly support the interface identifier, and there are no checks to ensure this. As a result, the ABI for a particular implementation may not match its trait, violating assumed invariants within Rust and potentially leading to failed contract integrations.

Consider refactoring the implementation of the `interface_id` macro to verify compatibility in custom selectors with respective trait implementations.

Update: *Acknowledged, will resolve. The Contracts for Stylus team stated:*

> *We have recommended the Stylus SDK team to include the `#[interface_id]` functionality and the option to retrieve selectors of each method and auto-implement each router function to build inheritance this way (Doing so would allow compile-time checks on custom selector differences between traits and implementation). We will work with the team to add this functionality in future iterations.*

Low Severity

L-01 Verification Functions for Custom Hashing in Merkle Trees Assume Commutative Hashing

The functions verify_with_builder and verify_multi_proof_with_builder serve as verifiers for Merkle trees with custom hashing functions. However, these functions reconstruct the Merkle root in a commutative manner, which assumes that all custom hashing algorithms employ commutative hashing. Although commutative hashing is common practice, this assumption cannot be guaranteed for all custom hashing processes.

Consider updating the documentation to specify that the Merkle tree hashing process must be constructed commutatively when using custom hashing algorithms.

Update: *Resolved in pull #674 at commit 6f75f80.*

L-02 Inconsistent Data Types for `carry` and `borrow` Parameters

The arithmetic carry (for addition) and borrow (for subtraction) parameters are defined as the `Limb` type in the `adc` and `sbb` functions, but as the `bool` type in the `adc_assign` and `sbb_assign` functions. These parameters are currently expected to hold only the values `0` or `1`, as required by the functions that utilize them. However, unintended behavior may arise if a function erroneously passes a carry or borrow parameter with a value greater than 1.

For example, the overflow protection mechanism will fail if `a = 0`, `b = u64::MAX`, and `borrow = 2` in the `sbb` function. Consider adopting the `bool` type exclusively for carry and borrow parameters, casting them to `Limb` during calculations as necessary.

Update: *Resolved in pull #675 at commit cde91fc.*

L-03 Encoding Ambiguity in `ct_num_bits` Function

According to the comment of the `ct_num_bits` function, it returns the minimum number of bits required to encode a given number. Currently, it returns `0` for the number `0`. However, in practical encoding scenarios, a minimum of **one bit** is necessary to represent any number, including `0`. Returning `0` bits for the number `0` may introduce ambiguity or errors in systems that expect at least one bit to represent valid numbers. Consider revising the function to ensure it returns at least one bit for all inputs.

Update: *Resolved in pull #681 at commit e36c292.*

L-04 Absence of Overflow Check in `from_str_hex` Function

The `from_str_hex` function does not include an overflow check when parsing hexadecimal strings into a `Uint<LIMBS>` type. This omission may result in an unexpected index out-of-bounds panic if the input string represents a number exceeding the capacity of the `Uint<LIMBS>` type. In contrast, the `from_str_radix` function incorporates an overflow check within the `ct_add` function, ensuring that overflow conditions are explicitly detected and trigger a panic with a clear error message.

Consider implementing an overflow check in the `from_str_hex` function to enhance its robustness.

Update: *Resolved in pull #673 at comit a988e2b.*

L-05 Underlying Decimals in `Erc20Wrapper` Can Be Set Incorrectly

The `Erc20Wrapper` contracts set the underlying token address during construction but do not set the `underlying_decimals` storage field from the decimals of the underlying token. The underlying decimals must then be set manually, which can lead to critical errors in token functionality if the decimals are set incorrectly. Consider assigning the `underlying_decimals` field in the constructor using the `decimals` of the underlying token, or including checks to ensure the assigned values are equal.

Update: *Resolved in pull request #699 at commit add8075.*

L-06 State Mutability Can Be Restricted in IErc4626

The `IErc4626` trait currently marks several methods with `&mut self` even though they only need to read contract storage, such as `total_assets`, `convert_to_shares`, `convert_to_assets`, etc.

By requiring `&mut self` on methods that are conceptually read-only, the trait allows implementations to introduce unintended or malicious state changes. This is because a developer could override a “read-only” function and write to storage. To preserve expected invariants and prevent misuse, consider changing the signature of any `IErc4626` method that does not modify state to take `&self` rather than `&mut self`.

Update: *Resolved in [pull request #698](#) at commit [33b15ed](#).*

L-07 Custom Selectors in Trait With `interface_id` Allows Unreachable Patterns

The `#[interface_id]` attribute enables custom selectors to be added to function signatures in traits via the `#[selector]` attribute. This mirrors the functionality available for functions within `#[public]` implementation blocks in the Stylus SDK. Code that includes functions with identical selectors in `#[public]` blocks yields an unreachable pattern error; however, this enforcement is not present in traits with the `#[interface_id]`. Consequently, it is possible to define traits with the `#[interface_id]` attribute that can never be implemented. To maintain expected programming invariants and reduce developer errors, consider prohibiting the creation of traits that have duplicate function selectors.

Update: *Resolved in [pull request #702](#) at commit [b68f7ae](#).*

L-08 Missing Trait Components in Generated Code From `interface_id` Macro

The `#[interface_id]` macro processes a trait by adding an `interface_id` function to the trait's [re-generated code](#), maintaining the original content apart from this intentional addition. However, two critical components are omitted:

1. **Generic bounds are missing:**

The `ty_generics` variable expands only to the generic type name without including bounds. For example, a trait like `trait Foo<T: U>` would expand to `trait Foo<T>`. This omission can allow implementers to create contracts with generic types that do not respect the specified bounds, potentially violating intended invariants.

1. **`unsafe` qualifier is lost:**

Rust allows traits to be declared with the `unsafe` qualifier, indicating that implementations may require the use of unsafe features. The loss of this keyword can silently weaken the safety guarantees intended by the `unsafe trait`, allowing implementations to compile without the necessary flag.

When re-generating a trait in the `interface_id` macro, it is advisable to include all declaration components, ensuring that the output precisely replicates the source trait. This practice prevents implementers from bypassing generic bounds or omitting critical annotations, thereby preserving both type safety and intended invariants.

Update: *Resolved in pull request #705 at commit 184bf10.*

L-09 Potential Vulnerability in Big Integer Library to Timing Side-Channel Attacks

Certain functions like ct_ge(), ct_gt(), and cmp() in the big integer cryptographic library utilize early returns when comparing values. The use of early returns does not ensure constant-time execution across all inputs, potentially exposing the library to timing side-channel attacks. When the input values contain sensitive information, such attacks could enable an attacker to analyze the execution time to infer details about those values.

Consider accumulating the result and returning it at the end of the loop or using bitwise operations to ensure constant-time execution.

Update: *Resolved in pull #700 at commit 7ae1021. The fix reversed the comparison sequence and made the execution time constant.*

Notes & Additional Information

N-01 Consistent Application of the #[must_use] Attribute in Merkle Proof Verification Functions

The `#[must_use]` attribute enables developers to annotate a function, ensuring its result is not ignored. In the Merkle module, the verification function `verify` for a single leaf is annotated with the `#[must_use]` attribute. However, the verification function `verify_multi_proof` for multiple leaves lacks this annotation, potentially leading to inconsistent usage.

Consider applying the `#[must_use]` attribute consistently to both functions or documenting the difference.

Update: *Resolved in pull #708 at commit a15df81.*

N-02 Redundant Attribute in the `adc` Function

The `adc` function includes a function-level attribute, `#[allow(clippy::cast_possible_truncation)]`. However, a file-level attribute at the top of the file already applies this setting across the entire file, rendering the function-level attribute redundant. It is recommended to remove the function-level attribute from the `adc` function to enhance code clarity and maintainability.

Update: *Resolved in pull #706 at commit 19974a9.*

N-03 Absence of `#[inline(always)]` Attributes in Certain Methods

Several `Uint` methods, including `ct_lt`, `ct_is_zero`, and `ct_eq`, lack the `#[inline(always)]` attribute, in contrast to similar methods such as `ct_ge`, `ct_gt`, and `ct_le`. These methods are likely employed in performance-critical contexts. Applying the `#[inline(always)]` attribute would ensure consistent inlining behavior, minimize function call overhead, and enhance performance. Consider adding the `#[inline(always)]` attribute to these methods.

Update: *Resolved in pull #707 at commit 80402e2.*

N-04 Incorrect Comment in `PartialEq` Implementation for Projective Points

The comment for the `PartialEq` implementation of `Projective` points states that points `(X, Y, Z)` and `(X', Y', Z')` are equal when `(X * Z^2) = (X' * Z'^2)` and `(Y * Z^3) = (Y' * Z'^3)`. However, the implementation verifies `(X * Z'^2) = (X' * Z^2)` and `(Y * Z'^3) = (Y' * Z^3)`, which is correct. The comment is inaccurate because it fails to reflect the correct order of operations in the implementation. Consider revising the comment.

Update: *Resolved in pull #711 at commit 12adbbc.*

N-05 Source Code Attribution Concerns

Certain components of the library, such as the curve module, which incorporates code derived from algebra/ec, and the field module, which utilizes code from algebra/ff, include portions of code from external codebases without proper attribution. It is recommended to include references to the original codebases within the code comments to ensure proper attribution.

***Update:** Resolved at pull #712 at commit eaa0ba2.*

N-06 Documentation Enhancements

Two opportunities for enhancing documentation were identified within the codebase:

- The absorb function enforces that the mode is set to `Absorbing`. However, the squeeze function permits the mode to remain `Absorbing`, without a reciprocal check to ensure the mode is `Squeezing`. This behavior is correct, as the function should remain in `Squeezing` mode until reinitialized.

To enhance clarity, consider adding a comment to the absorb function: "*Transitions from Absorbing to Squeezing mode are unidirectional.*" This comment emphasizes that once in `Squeezing` mode, returning to `Absorbing` mode is not possible, thereby justifying the if-condition. Similarly, consider adding a comment to the squeeze function: "*When invoked from Absorbing mode, this function triggers a permutation and transitions to Squeezing mode.*"

- The Poseidon2 implementation forms part of a low-level library, providing foundational components for constructing a complete hash function. Consequently, high-level functionalities, such as padding, domain separation, and absorb/squeeze transitions, are the responsibility of the library user.

To clarify this, consider amending the documentation with a statement such as: "*This interface does not implement padding or domain separation. Users are responsible for padding inputs, prepending domain separation tags, and managing absorb/squeeze transitions.*"

Consider implementing the recommended documentation enhancements to improve clarity and usability.

***Update:** Resolved in pull #710 at commit eb72ebc.*

N-07 Inconsistent Behavior of Shift Overflow Compared to Rust Native Integer Types

As specified in Rust RFC 560, for a type of width `N` (for example, `u32` where `N = 32`), if the right-hand operand (shift amount) is greater than or equal to `N`, it is masked to `shift_amount % N`. This ensures the effective shift amount remains within the valid range. For example:

- Shifting `1u32 << 32` is equivalent to `1u32 << 0` (since `32 % 32 = 0`), resulting in `1`.
- Shifting `1u32 << 33` is equivalent to `1u32 << 1` (since `33 % 32 = 1`), resulting in `2`.

However, the implementation of the `shr_assign` and `shl_assign` functions returns `Zero` for overflow shifts. This behavior is inconsistent with the default operation of native integer shift operations in Rust. Consider aligning the implementation with Rust's native behavior to ensure consistency and predictability.

Update: *Resolved in pull #658 at commit c38b883. The shift operation will revert for shift overflow.*

N-08 Misleading Documentation

The `#[interface_id]` procedural macro is described as adding an associated constant [1, 2], but it is actually added as a function. Consider correcting this comment to reflect accurate behavior.

Update: *Resolved partially in pull request #705 at commit 184bf10 and partially in pull request #702 at commit b68f7ae.*

N-09 Typographical Errors

Throughout the codebase, the following typographical errors were identified:

- The variable "`underline_token`" should be renamed to "`underlying_token`."
- The test "`prevents_non_onwers_from_transferring`" should be renamed to "prevents_non_owners_from_transferring."

- In line 111 of `contracts/src/access/control.rs`, the comment should say, "The caller of a function..."

Consider correcting these errors and any others within the codebase to improve overall clarity.

Update: *Resolved in pull request #709 at commit 917b9e2.*

N-10 Insufficient Unit Test Coverage

Certain parts of the codebase lack adequate unit tests. For example, the crypto arithmetic limb.rs module does not exercise all of its functions. Consider increasing test coverage to ensure correctness across edge cases.

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

> *We will improve test coverage both on crypto and contracts part this during our Milestone 3.*

Recommendations

Recommendations

- **Check for Poseidon2 S-box Invertibility**: Add a check `gcd(P::D, F::MODULUS - 1) == 1` in the `new()` function to ensure the invertibility of the S-box mapping `f(x)=x^D`.

- **Enforce Even Number of Full Rounds and Minimal Number of Rounds in Poseidon2**: According to the Poseidon2 paper, the total number of full rounds must be an even number. The current implementation assumes this condition is met. Consider adding a check to ensure that this requirement is enforced. Also, a minimal number of rounds must be done to achieve the required security parameter.

Conclusion

The Contracts for Stylus library has been upgraded to leverage the latest Stylus SDK and now offers a richer, more user-friendly feature set. Notably, it adds a fully integrated ERC-4626 vault for ERC-20 tokens—an essential building block for DeFi applications—and expands ERC-1155 support with burnable extensions, metadata URI management, and supply controls, addressing growing NFT use cases. These enhancements significantly broaden the library's applicability and demonstrate a clear, well-organized code structure.

The Crypto Library offers a comprehensive suite of cryptographic utilities across six modules, designed for secure and efficient blockchain applications. It features a generic hash module for producing larger-size outputs, a merkle module for efficient Merkle proof verification, and an arithmetic module for safe big integer operations. Additionally, the curve module enables elliptic curve operations with affine and projective representations, while the field module supports finite field arithmetic computations. The Poseidon2 and Pedersen modules provide fast, blockchain-optimized hash functions, tailored for zero-knowledge proofs.

The audit identified one high-severity issue and two medium-severity issues, along with several low-severity findings and general notes across the contracts and crypto library scopes. To maintain robust security and feature integrity, regular audits are strongly recommended whenever major library updates are introduced. Continued development and iteration will further strengthen the library's utility and contribution to the Arbitrum Stylus ecosystem.