OpenZeppelin | security

Arbitrum Stylus SDK v0.10 Audit

December 10, 2025

Table of Contents

Summary

Type	SDK
Timeline	From 2025-10-20 To 2025-11-07 From 2025-11-24 To 2025-11-28
Languages	Rust

Total Issues	50 (7 resolved, 2 partially resolved)
Critical Severity Issues	1 (1 resolved)
High Severity Issues	3 (3 resolved)
Medium Severity Issues	19 (3 resolved, 1 partially resolved)
Low Severity Issues	17 (0 resolved, 1 partially resolved)
Notes & Additional Information	10 (0 resolved)

Scope

OpenZeppelin audited the OffchainLabs/stylus-sdk-rs library at commit 4003c3f (v0.10-rc.1). This is a follow-up to the v0.9.0 release, focusing on the refactors and newly added features.

In scope are all the changes made in the following directories:

```
├── cargo-stylus/**/*.rs
├── stylus-tools/**/*.rs
├── stylus-core/src/**/*.rs
├── stylus-proc/src/**/*.rs
└── stylus-sdk/src/**/*.rs
```

Subsequently, a diff audit was performed between commit c79fa8b and commit 4003c3f. For this audit, the scope was expanded to include `cargo-stylus/**/*.rs` and `stylus-tools/**/*.rs` as well.

System Overview

Stylus SDK

The changes under review introduce a series of improvements, including breaking changes, new features, and important bug fixes that expand the library's scope and usability.

A refactor of the Stylus SDK included the unification of host interactions through a single `VM` struct delegating to an updated `Host` trait, eliminating global helpers. Cross-contract calls and deployments were overhauled with a type safe `Call` builder alongside standalone functions. ABI generation was modernized with the addition of `SELECTOR_ABI` encodings and `abi_encode_return` methods to fix selector mismatches and tuple encoding issues, and ensure precise Solidity alignment. A high-level event emission helper was added to existing low-level logging functionality.

Storage types now expose `HostAccess` implementations and correct borrow lifetimes, with vectors and maps receiving safety cleanups and more clear mutability rules. Explicit wrapping and checking arithmetic helper functions were added, and ABI encoding has been aligned with the `AbiType` interface which replaced old SolType based paths, standardizing how return data is produced.

The procedural macros have been updated to reflect these structural changes, including macros generating method selectors, ABI encoders/decoders, and purity metadata using the SDK's own `AbiType` abstraction instead of SolType-based helpers. Generated entry points now operate through the `VM` object, aligning with the refactored host interactions model throughout the codebase.

Cargo Stylus

The codebase has been restructured from a monolithic repository into a modular architecture with two main components: `cargo-stylus` (CLI interface and command parsing) and `stylus-tools` (core functionality for building, deploying, and managing contracts). The CLI now supports Cargo workspaces, allowing developers to build and deploy multiple contracts within a single workspace structure, with contracts marked by a `Stylus.toml` configuration file that will support per-contract and workspace-wide configuration options.

The `deploy` command has been significantly enhanced: it now supports contracts with constructors, including `payable` constructors, via `--constructor-args` and `--constructor-value` flags. Deployment now uses a unified `StylusDeployer` contract that handles deployment, activation, and constructor initialization. The `verify` command now explicitly supports Docker-based reproducible builds, ensuring local builds match deployed bytecode. While this improves verification reliability, it introduces Docker as a dependency and requires a review of the Docker image management, build reproducibility, and potential supply chain risks.

Security Model and Trust Assumptions

The system now assumes that any provided `Host` implementation is correct and that callers will enforce their own contract-level invariants when using low-level call or deployment APIs.

The SDK defines no privileged roles. As such, trust assumptions must be designed at the application layer.

Critical Severity

C-01 Contracts Can Never Be Verified

The `verify` function determines whether a deployment transaction includes a constructor call by decoding the transaction input via `deployCall::abi_decode`, expecting the `deploy` selector as the first four bytes. However, this approach only works for contracts deployed through the Stylus deployer contract, which wraps constructor calls.

Contracts without constructors can be deployed directly using a standard `CREATE` call, in which case the transaction input will not follow the expected ABI format. As a result, `abi_decode` will always for such deployments. Even when a constructor is detected, verification will still fail because `verify_constructor_deployment` is currently unimplemented. In practice, this means that the `verify` function fails for all deployment types, both with and without constructors.

Consider refactoring the verification logic to correctly handle contracts that do and do not use Stylus constructors for deployments.

Update: *Resolved at commit cc54d33.*

High Severity

H-01 `get-initcode` Command is Non-Functional

The `cargo stylus get-initcode` command is intended to generate and print the initcode for a contract. However, it is currently non-functional. The command's `exec` function in `get_initcode.rs` panics when the `--output` flag is omitted. The documented behavior is to default to stdout, but the implementation contains a `todo!()` macro in the `None` branch.

The execution then proceeds to call the `ops::write_initcode` function, with or without the `output` argument. This underlying function, which is responsible for building the WASM, compressing it, and generating the initcode, has its entire implementation commented out.

As a result, even if the command were to execute, it would perform no action and simply return `Ok(())`, producing an empty output. These two issues make the `get-initcode` command unusable. Users attempting to use it will either encounter a panic (if no `--output` is specified) or get an empty file, contrary to the feature's purpose.

Within `initcode.rs`, consider implementing logic to correctly build, compress, and write the initcode. In addition, consider replacing `todo!()` in `get_initcode.rs` with logic that uses `std::io::stdout()` as the writer when the `--output` flag is not provided.

Update: *Resolved at commit 4b70765.*

H-02 Check and Deploy Ignores `--features` Flag Producing Incorrect Bytecode

The `deploy` subcommand constructs its configuration via `DeployArgs::config`, which takes a `CheckArgs` and discards `BuildArgs`. `CheckArgs::config` returns a `CheckConfig` with a default `BuildConfig`, leaving the features list empty. At the same time, `CheckConfig` includes a `build` field consumed by `check_contract` and `build_contract`, which is used, for example, in `cargo stylus check`. However, the check command is built from `ActivationArgs` only, resulting in the same default build config. As a result, deployment and check builds ignore CLI features and produce default-feature bytecode. This leads to incorrectly deployed bytecode for contracts that are expected to be deployed or checked with any features turned on.

Consider threading `BuildArgs.features` into `CheckConfig.build` for the build step during deployment and contract checking.

Update: *Resolved at commit 8534421.*

H-03 `required_slots` Undercounts After Packed Bytes Followed By a Multi-Word Field

The `#[storage]` macro computes storage layout via two divergent paths:

- `init`: When placing a multi-word field (e.g., `StorageArray`), it first aligns to a new slot if the remaining intra-slot space cannot fit the field's byte-sized header, and then reserves the field's full slot count.
- `size`: For multi-word fields, it skips this pre-alignment check, directly adding the field's slot count to the total without accounting for partial intra-slot space.

This mismatch causes `REQUIRED_SLOTS` to undercount slots when a small "packed" field (e.g., `StorageBool`) precedes a multi-word field. In nested layouts, parent structs will then rely on the undercounted slot count of child structs, placing subsequent fields one slot too early. This creates slot overlap and persistent storage corruption.

Consider modifying the compile-time `size` logic to mirror the runtime `init` logic for multi-word fields. Specifically, the branch that handles multi-word fields should first perform the same byte-packing alignment check that the runtime does, advancing to a new slot if the field's header does not fit in the remaining space. This alignment should be checked before adding the field's full slot count, which will ensure that the calculated `REQUIRED_SLOTS` constant accurately matches the actual storage slots consumed at runtime.

Update: *Resolved at commit 646c545.*

Medium Severity

M-01 `hash_project` Function Returns Default Hash, Undermining Build Integrity

The `hash_project` function, responsible for generating a unique hash representing a project's source code, is currently incomplete. Instead of computing a hash based on the project's actual source files and configuration, it returns a default hash. This value is used by downstream processes, including the `cargo stylus check` and `cargo stylus verify` commands, which relies on the `ProjectHash` for build reproducibility, source code verification, and artifact integrity. The resulting hash is embedded into the compiled WASM binary as a custom section named `project_hash`. Since `hash_project` does not perform real hashing, every project produces the same hash, regardless of its content.

The `cargo stylus check` command builds the WASM and calls the `process_wasm_file` function, which embeds the `project_hash` into the binary. When run without a specified WASM file, it calls the `hash_project` function and receives a default hash. When run with `--wasm-file`, it explicitly uses `ProjectHash::default()`, also resulting in a default hash. In both cases, the checked artifact is stamped with an incorrect, non-unique hash. This undermines the integrity of the check command and the validation pipeline.

Consider implementing proper hashing in `hash_project` to compute a deterministic hash of all relevant project sources, configuration files, and dependencies.

Update: *Resolved at commit 5e6cfa3.*

M-02 Docker Helpers Skip Exit-Status Checks

The `wait()` and `run()` functions in `docker/cmd.rs` treat any `wait()` return as a success, unless there is an OS-level spawn or wait error (e.g., docker is not running). These function do not check the child process's exit code. As a result, docker commands that fail within the child process (for instance, exit code 1) are still reported as `Ok(())`, causing the downstream processes, such as reproducible build and verification, to appear successful even when the underlying Docker invocation failed.

Consider checking the `ExitStatus` of the child process, and in case of failure, returning a relevant error message.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> We have not encountered any issues with this due to our limited and consistent use of docker commands. We have noted this suggestion if we rework our docker usage and wish to add the extra sanity checks.

M-03 Unaddressed `todo!()` Instances

In addition to the existing `todo!()` placeholders that leave the initcode and verification features incomplete, there are several other parts of the codebase where `todo!()` is used in place of the expected implementation:

- In `abi.rs`
- In `solc.rs`, here and here
- In `frame.rs`
- In `git.rs`

Consider replacing all instances of `todo()!` with the appropriate implementation to avoid unexpected panic scenarios during normal usage of the CLI tool.

Update: *Partially resolved in PR 372 and at commit 347439d.* There remains an unaddressed `todo!()` in `frame.rs`.

M-04 Project Hash Is Stripped During Wasm Processing

In the `process_wasm_file` function, which is used during contract checking and deployment, the project hash is added as a custom section to the WASM binary for use in reproducible build verification. However, all custom sections, including this project hash, are later removed by the `strip_user_metadata` function. As a result, the project hash metadata is never actually included in the deployed bytecode, potentially weakening the verification process. Any future verification feature relying on the embedded project hash will not work with contracts deployed using the current cargo stylus workflow.

Consider adding logic to detect the project hash section and skip removing it in the `strip_user_metadata` function.

Update: *Resolved at commit cae943d.*

M-05 Inflexible Deployment Prelude Can Lead to Unverifiable Contracts

The deployment prelude is currently hardcoded, making it inflexible for verifying contracts that may have been deployed with a different prelude. Since `verify` checks the deploy transaction calldata against locally built WASM from the current SDK, this can fail for contracts that were deployed using older versions of `cargo stylus` that appended a hash after the version. It is also possible that future versions may change the version byte or modify the existing prelude.

Consider refactoring verification to remain flexible with older or updated versions of the deployment prelude.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Having the required static prelude is necessary for our current verification strategy. So far we have not been made aware of any desire for custom contract preludes, but we can explore that if a use-case is brought to our attention.

M-06 Verification Failure Due to Bytecode Mismatch Silently Returns Success

The `verify` function calls `verify_create_deployment`, when the contract being verified does not have a constructor, which returns `VerificationStatus::Failure(VerificationFailure)` in case of a mismatch between the deployed and locally built codes. However, this is not propagated as an error and `verify` function returns `Ok(())`, mapping to exit code 0. Hence, any attempted verification that fails due to bytecode mismatch will report no error, which can lead to users believing that their contract was successfully verified.

Consider explicitly handling `VerificationStatus` in the CLI: print details on `Failure` and return a non-zero exit status so that mismatches can be properly detected and handled.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Currently investigating.

M-07 Missing Transaction Receipt Status Validation in Cache and Activation Operations

The `place_bid` and `activate_contract` functions fail to validate transaction receipt status after sending on-chain transactions, resulting in false success reporting when transactions revert. Both functions call `.get_receipt().await?` without subsequently checking `receipt.status()`, unlike the correctly implemented `DeploymentRequest::exec` which explicitly validates status and returns `DeploymentError::Reverted` on failure. While pre-execution simulations using `.call()` provide some protection, they cannot guarantee transaction success due to state changes between simulation and execution. This can lead users to believe that operations succeeded when they consumed gas without achieving the intended state change, and may result in repeated failed attempts without proper error feedback.

Consider checking status receipts and handling errors on failure.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> We have currently not encountered any issues without the additional checks, but we have made a note to ensure error handling is more robust in the future.

M-08 Reproducible Verify Omits `--deployment-tx` Flag Causing CLI Parse Failure

The `verify` command in reproducible mode (default) constructs invalid CLI arguments when passing transaction hashes to the Docker container, causing verification to fail silently. The verify command only accepts `--deployment-tx` as a named flag, causing clap to reject the hash as an unexpected positional argument. This results in immediate command-line parsing failure inside the Docker container. Due to a separate bug where `wait()` succeeds, but the exit status is never checked, this parsing failure returns `Ok(())` to the caller, making the verification command appear successful when it actually failed. Users running `cargo stylus verify` without `--no-verify` (the default path for reproducible verification) do not receive any error feedback and incorrectly believe their deployed contracts have been verified, potentially leading to unverified contracts being used in production.

Consider adding the expected CLI arguments using `cli_args.push(String::from("--deployment-tx"))` before pushing the hash value, and separately checking `status.success()` after `wait()` in the Docker run function.

Update: *Resolved at commit d59a5c0.*

M-09 `account_code` Returns Box Pointer Size Instead of Actual Code Length

The `account_code` function copies the returned code bytes with the two-argument `copy!` macro, which computes the copy length as `mem::size_of_val(&src)`. Since `code` is a `Box<[u8]>`, this copies only the boxed fat-pointer size (typically two words) while the function reports `code.len()` as the number of bytes written. The result is truncated writes and inconsistent return lengths. This behavior corrupts local replay/debugging: consumers trust the returned length and read uninitialized memory beyond what was actually written. Other paths correctly use the three-argument form with the real length (for instance, `read_return_data` uses `copy!(data, dest, data.len())`).

Consider switching `account_code` to copy `code.len()` bytes (the three-argument `copy!` form), or refactor the macro to reject unsized sources for the two-argument variant. This will ensure that the number of bytes written matches the returned length and preserve replay correctness.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> We are exploring the proposed solution here to avoid making a change which fails to consider all aspects of the current funcitonality.

M-10 Overload Disambiguation Suffixes Can Collide with User-Defined Function Names

The `c_gen` function in `codegen.rs` resolves function overloads by appending numeric suffixes (e.g., _1, _2) to create unique C identifiers. However, this mechanism can lead to a name collision if a user defines a distinct function with a name that matches a generated suffixed name, for instance, defining functions `f()` and `f(uint)` alongside a separate function named `f_1()` results in duplicate `SELECTOR_f_1` macros and multiple router branches comparing against the same selector.

Consequently, this may cause compile-time errors due to macro redefinition. If the compilation succeeds by accepting the last macro definition as the correct one, it may lead to incorrect function routing at runtime, where calls intended for one function are dispatched to another, potentially causing inconsistent payability checks and unexpected behavior.

Consider enforcing global uniqueness of generated C identifiers by reserving the numeric suffix space such as mangling literal names that end with a numeric suffix or by adopting a disambiguation scheme that cannot collide, such as hashing full function signatures. In addition, consider introducing static assertions or generator-time errors when a collision is detected to prevent the generation of ambiguous routers.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Generation of C bindings for Stylus should be revisited at a later release. These suggestions will be taken into consideration at that time.

M-11 `run_reproducible` Panics on Local Crates

The `run_reproducible` function in `reproducible.rs` utilizes `package.source.unwrap().repr` to construct the host path for Docker bind mounts. For local crates, `package.source` is `None`, as these crates are sourced directly from the local filesystem and not from an external registry or git repository. When the code attempts to call `.unwrap()` on this `None` value, it triggers a panic, causing the `cargo stylus` process to crash immediately.

Consider using `package.manifest_path.parent()` as the host path for handling cases such as the aforementioned.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> We are exploring the proposed solution here to avoid making a change which fails to consider all aspects of the current funcitonality.

M-12 Missing Contract Selection Propagation in Reproducible Verification

When performing contract verification with the reproducible Docker container, the `--contract` selection is not forwarded to the inner CLI invocation. `cargo stylus verify` spawns a reproducible containerized run when verification is enabled. The argument list is built as ['verify', '--no-verify'] and only the raw transaction hash is appended. The `--contract` selections are not propagated.

Consequently, inside the container, the CLI recomputes the contract set using `ProjectArgs::contracts()`, which defaults to `workspace.default_contracts()` when no `--contract` is provided. In multi-contract workspaces, this can lead to the verification of additional or unintended contracts within the container, resulting in misleading verification outcomes and wasted CI time.

Consider forwarding every user selection that affects contract resolution into the reproducible invocation. For example, include a `--contract <pkg>` flag per selected package in the `cli_args` that are passed to the container, and ensure that the inner CLI receives the exact same contract set. Adding tests that assert the inner/outer selections match would further reduce regression risk.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Currently investigating.

M-13 C Router Ignores Fallback/Receive Payability Semantics

The C router generated by `c_gen` function dispatches short calldata and unknown selectors to a default handler, forwarding `msg.value` and without any awareness of ABI fallback/receive entries. Specifically, the router template retrieves `msg_value` and, if `len < 4`, it

returns `default_func`. However, on selector mismatch, it falls through to `default_func` again. In contrast, for matched functions the generator emits a non-`payable` guard that reverts on non-zero value. The generator only iterates `abi.functions()` and does not account for ABI fallback or receive entries, nor does it synthesize dedicated handlers.

As a result, when a contract's ABI expects a revert for non-`payable` fallback/receive, the generated router may accept value and invoke `default_func` instead. This diverges from EVM payability semantics and can lead to silent ETH acceptance in projects that rely on the generator's default path without adding explicit guards in `default_func`.

Consider extending the generator to handle fallback vs receive semantics explicitly and to enforce a `value == 0` check on the default dispatch path when the ABI indicates non-`payable`. Alternatively, consider synthesizing distinct receive and fallback stubs based on ABI metadata and insert reverts for unsupported paths. If it is intended to keep a generic `default_func`, consider emitting a generated check that reverts on a non-zero value unless the contract declares a `payable` fallback.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Generation of C bindings for Stylus should be revisited at a later release. These suggestions will be taken into consideration at that time.

M-14 C Bindings Can Be Generated With Invalid Header

The C header generator constructs a preprocessor guard using a `unique_identifier` formed by capitalizing and concatenating the Solidity file name and contract name, with no character filtering. This occurs when building the identifier and emitting it into `#ifndef`/`#define`/`#endif` lines. Since the Solidity file name is sourced from the JSON `contracts`, it commonly contains characters such as `/` and `.` that are not valid in C identifiers. As a result, generated headers may fail to preprocess or compile due to an invalid macro name in `#ifndef`.

Consider sanitizing the header guard to `[A-Z0-9_]`, for example, by mapping invalid characters to `_` and validating that input keys match expected patterns before generation.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Generation of C bindings for Stylus should be revisited at a later release. These suggestions will be taken into consideration at that time.

M-15 Negative Mapping Keys Can Derive Incompatible Solidity Slots

In `storage_sdk::storage`, `StorageMap<K,V>` derives element slots as `keccak256(h(key) || root)`, where `h` is a function that is applied to the key depending on its type:

- For value types, h pads the value to 32 bytes in the same way as when storing the value in memory.
- For strings and byte arrays, `h(k)` is just the unpadded data.

The SDK encodes signed keys by left-padding with zeros. Primitive signed integers follow the same pattern via an unsigned cast in impl_key!. This produces a 32-byte, zero-extended representation for negative values with bit-width < 256.

For Solidity storage compatibility, signed integers must be sign-extended to 32 bytes in the preimage. For example, `int8(-1)` should hash `0xff..ff || slot`. However, but the current SDK encodes `0x00..00ff || slot`, yielding a different hash. This causes cross-language storage divergence for negative keys and can break migrations or multi-language access patterns and external tooling relying on the Solidity layout. Within the SDK, reads/writes remain self-consistent, but interop with Solidity will target different slots.

Following the Solidity encoding specifications, consider sign-extending signed keys that hold less than 256 bits to 32 bytes before hashing to match Solidity's storage encoding. Doing so will avoid discrepancies between the storage layouts of both languages.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> We currently attempt to be as close as possible to Solidity storage, but there are instances where we are comfortable diverging. The case reported here will be investigated and either documented as different, or changed. The biggest consideration on our side is the avoid making a breaking change from stylus -> stylus which would harm upgradability.

M-16 Zero-Sized Structs and Arrays in Storage Cause Layout Misalignment

Within the `stylus_proc::macros::storage` module, the logic for calculating `REQUIRED_SLOTS` incorrectly handles zero-sized types, causing them to occupy one full storage slot.

This issue manifests in two related ways:

- **Unit/Empty Structs**: The storage macro computes `REQUIRED_SLOTS = 1` for empty structs. In the parent's `init` and `size` logic, this empty child causes a slot bump and resets packing space.
- **Zero-Length Arrays**: The `StorageArray<T, 0>` struct correctly defines `REQUIRED_SLOTS` as 0, but inherits the 32-byte `SLOT_BYTES` default. The storage macro's packing logic subtracts these 32 bytes when `REQUIRED_SLOTS == 0`, which is visible in the macro's initializer and sizer.

As a result, both zero-length `StorageArray<T, 0>` and empty structs consume 32 bytes in their parent struct, creating a layout hole and shifting subsequent fields by one slot. This breaks Solidity compatibility, diverges from expectations when mirroring Solidity layouts, and can silently corrupt storage.

The empty slot is never accessed, yet the physical span of the parent increases, which can desynchronize Stylus and Solidity storage roots and break upgrades. For example, when upgrading a proxy contract's implementation from Stylus to Solidity, developers must take into account empty fields, replacing them with 32-byte dummy fields to avoid storage misalignment.

Consider disallowing empty storage fields and rejecting zero-length fixed arrays at parse time. This would resolve the ambiguity, as these types do not hold any information and cause confusion. This change guarantees that the required slots are accurately computed at compile-time to reflect the runtime storage layout and restores compatibility with Solidity, allowing developers to migrate between languages and integrate proxy contracts without facing unexpected issues.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Usage of these patterns is not supported. We will be updating our documentation to reflect this, and implemented proper error reporting at a later date.

M-17 Arguments of Tuple Type Render Incorrect Function Selectors and ABI

Within the `stylus_proc::types` module, the `sol_interface` pipeline converts Solidity types to a pair of Rust and Solidity representations. The problem arises when function arguments are of the Rust tuple type, leading to incorrectly generated function selectors and ABIs. For single-element tuples, `SolidityTypeInfo::from` has a special case that returns

the inner type entirely. As a result, the stored `sol_type` for a parameter declared as a single-element tuple holds the inner type without parentheses. The selector preimage is then constructed from `sol_type.to_string()`, making the function signature indistinguishable from one declared with a bare inner type parameter and computing an incorrect selector.

For empty and multi-element tuples, another issue arises. The function selectors are exported semantically correct as tuples. However, this syntax is then rendered directly into the exported Solidity interface. This makes it impossible for Solidity contracts to use the exported interface since it will fail to compile. The current handling of tuple types in function arguments is inconsistent and incorrect, leading to ABI- and selector-generation issues. Single-element tuples are improperly unwrapped, causing selector collisions, while empty and multi-element tuples are rendered with syntax that is not valid in Solidity.

Consider resolving these discrepancies to ensure the SDK generates correct function selectors and produces a Solidity-compatible ABI that can be compiled and used for cross-language interoperability.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Generation of Export ABI for Stylus should be revisited at a later release. These suggestions will be taken into consideration at that time.

M-18 Multiple Mutating `Call` Accessors Can Exist Simultaneously

The `stylus_core::calls` module offers a high-level, type-safe builder for configuring calls to external contracts. The central `Call` struct uses constant generics and configuration methods to define a call's parameters. The main `Call` struct uses const generics and builder methods to set a call's parameters. The `StaticCallContext`, `NonPayableCallContext`, and `MutatingCallContext` traits then use this configuration to provide compile-time safety.

Within the `stylus_core::calls` module, `MutatingCallContext` is intended to enforce that external calls only occur while a unique `&mut TopLevelStorage` borrow is active, preventing reentrancy attacks. However, the mutating call types both immediately discard the borrow within their constructors. As a result, the `Call` object then carries no lifetime which ties it to the storage reference. Since `Call` also derives `Clone`, user code can create a mutating context once and continue to use it even after obtaining a live `StorageGuardMut` to a storage slot. An external call can then execute even with an outstanding mutable guard - if the callee reenters then it may acquire another guard to the same slot through safe APIs, breaking the intended aliasing barrier.

Consider threading a lifetime from `&mut TopLevelStorage` into the mutating call context so that its use is statically restricted to the borrow window. Alternatively, consider redesigning the API so that external calls require fresh evidence of an active borrow at each call site.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> We are investigating this as a potential bypass of the reentrancy guarantees, however since all calls in the safe interface will appropriately flush cache, this may not be an issue.

M-19 `BuildArgs` Does Not Consider `source_files_for_project_hash`

The `BuildArgs` struct in `common_args.rs` defines a command-line flag `--source-files-for-project-hash`. This flag is intended to allow users to specify which source files should be included when calculating the `project_hash` for reproducible builds and verification.

However, the implementation of the `config()` function for `BuildArgs` ignores this flag. It only forwards the `--features` flag and returns a `BuildConfig` with default values for all other fields. Moreover, the `source_files_for_project_hash` flag should be a part of the `ProjectArgs` struct to align with the `source_file_patterns` flag in the `ProjectConfig` struct.

Consider moving the `source_files_for_project_hash` flag from `BuildArgs` to `ProjectArgs` and updating the `ProjectArgs::config()` function to use this flag to populate the `source_file_patterns` field of the `ProjectConfig` struct.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Currently investigating, fix will be submitted as part of cargo-stylus fix/review.

Low Severity

L-01 Missing Cap on `data_fee_bump_percent` in Activation Contract

The `activate_contract` function allows for adjusting the activation data fee through the `data_fee_bump_percent` parameter of the `ActivationConfig` struct. During the `estimate_gas` function call, this `data_fee_bump_percent` defaults to 20%. While this mechanism provides flexibility in accounting for network conditions and gas fees, the lack of a cap on the `data_fee_bump_percent` parameter introduces risk. A user could mistakenly input an excessively high value, resulting in an inflated data fee. This could lead to unusually high transaction costs, depleting funds, and potential transaction failures due to insufficient balance.

Consider implementing a cap on the `data_fee_bump_percent`. This will allow the system to maintain flexibility in adjusting for network conditions while preventing extreme user errors that could negatively impact transaction costs.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Stylus-tools is at its first release stage. We are taking all input into consideration to improve the APIs for future releases.

L-02 Insufficient and Inconsistent In-line Documentation

The codebase lacks proper in-line documentation, which hinders the readability and maintenance of the codebase. Additionally, there are instances where the documentation is informal, such as in lines 52-53 of `codegen.rs`.

Consider adding standardized, descriptive in-line comments, as per the Rustdoc convention, throughout the codebase. Doing so will improve code readability and consistency across modules.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Stylus-tools is at its first release stage. We are taking all input into consideration to improve the APIs for future releases.

L-03 Misleading Error Message

This error message in the `network.rs` contract suggests the user to downgrade to cargo stylus version 0.2.1 in order to connect to the deprecated Stylus testnet. However, this guidance is outdated, as the referenced testnet is no longer supported. Encouraging users to downgrade introduces confusion and may lead them to use obsolete tooling that is incompatible with current network environments.

Consider updating the error message to only indicate the fact that the reference testnet is deprecated.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Stylus-tools is at its first release stage. We are taking all input into consideration to improve the APIs for future releases.

L-04 Incomplete Deprecated Network Check

The `check_endpoint` function in `network.rs` is intended to prevent users from interacting with deprecated testnets. However, its implementation is incomplete as it only checks for the Stylus testnet network and not for the Arbitrum Goerli testnet. As such, if a user attempts to use a deprecated but unblocked endpoint like Arbitrum Goerli, the `check_endpoint` guard will pass. The tool will then fail at a later stage when trying to establish a provider connection. This results in a poor user experience.

Consider adding a robust check in the `check_endpoint` function to include a comprehensive list of known deprecated endpoints, allowing the user to fail early with a clear error message.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Stylus-tools is at its first release stage. We are taking all input into consideration to improve the APIs for future releases.

L-05 Improper Handling of Missing Docker Base Image During Reproducible Build

During the Docker image lookup, the code attempts to build a local image derived from a base image hosted on Docker Hub. When the `cargo_stylus_version` parameter is not explicitly

provided, the code falls back to `env!("CARGO_PKG_VERSION")`, which is embedded at compile time. For example, release candidates like v0.10.0-rc.1 results in attempting to pull offchainlabs/cargo-stylus-base:0.10.0-rc.1 as the base image.

The issue is that the code does not validate whether this base image actually exists on Docker Hub before attempting to build. When the specified version has not been published to Docker Hub, the command does not return a result, but completes successfully with `output.status = 0`. The codebase does contain a check via `image_exists()` to determine if the image is already cached locally, but there is no corresponding check to verify that the remote base image exists on Docker Hub before attempting the build. This causes the build to fail late in the process with an unclear error message instead of failing early with a helpful explanation.

Consider adding validation to check whether the base image exists on Docker Hub before attempting to build. Alternatively, consider providing a fallback mechanism to use the latest stable version when the requested version is unavailable.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Stylus-tools is at its first release stage. We are taking all input into consideration to improve the APIs for future releases.

L-06 Unused Code

Throughout the codebase, multiple instances of unused code were identified:

- The `ArbAddressTable`, `ArbAggregator`, `ArbGasInfo`, `ArbInfo`, `ArbRetryableTx` and `ArbSys` precompiles are unused. Consider removing them, as well as the accompanying interfaces to them.
- The `check_workspace` function is currently not being used for any command associated with `cargo stylus` and does not appear to be used anywhere in the codebase.
- The `codegen.rs` contract contains only copyrights and licensing information. There is no relevant lines of code, the contract is unused during the `codegen` command execution and should be removed from the codebase.
- The `check` function of `ProcessOutput` is unused, consider removing the entire implementation of this struct.
- The `deploy` function in `cargo_stylus.rs` currently constructs `deploy_args` by pushing `--private-key` and the raw private key value, and subsequently calls the `call_deploy` function. This `call_deploy` function constructs and run another "cargo stylus-beta deploy..." command where the `args` are consumed, thereby,

exposing this private key to sub-processes. While it is unused, its implementation can result in exposure of private-keys if developers use it or copy/integrate it in downstream code.

- The deploy command defines a `constructor_signature` flag but does not use it in configuration or deployment. The flag is parsed in the CLI (argument definition) yet is not forwarded into `DeploymentConfig` (construction, `DeployArgs::config`). In the core, deployment unconditionally calls `get_constructor_signature` and selects the path based on that result.
- The `ops` module publicly re-exports `verify`, but the implementation is a stub that ignores its inputs and returns `Ok(())`. As such, it should be removed.

Consider removing all instances of unused code to improve the clarity and maintainability of the codebase.

Update: *Partially resolved in PR 372. `ProcessOutput::check`, `ops::check_workspace`, the `constructor_signature` CLI flag, and `ops::verify` are still present and unused/misleading, and the Arb* precompile bindings remain unused internally.*

L-07 Prelude Mismatch Is Never Detected During Failed Verification

The `verify_create_deployment` function compares the calldata from a contract deployment transaction with the compressed WASM built locally for a contract. If they do not match, the prelude parts are first compared to detect a mismatch. However, the `tx_prelude` and `build_prelude` are both assigned the same value, which always results in `None` being returned in the `prelude_mismatch` part of the returned `VerificationFailure` error. This can cause confusion by hiding the true nature of failed verifications due to mismatched preludes.

Consider assigning the correct `deployment_data.prelude()` to the `build_prelude` variable.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Stylus-tools is at its first release stage. We are taking all input into consideration to improve the APIs for future releases.

L-08 Incorrect Output In Cache Status Reporting

The `status` function of `cache.rs` reports the minimum bids for different contract sizes along with the current cache and queue sizes by calling the `CacheManager` contract. It includes a check to determine whether the cache is at capacity by comparing `queue_size < cache_size`, and if this condition is true, it informs the user that bids of 0 are accepted.

However, this is misleading since it does not account for the size of the contract for which a bid is to be placed. `CacheManager` only accepts a bid of 0 when the queue size + size of the contract is less than the cache size. Since the `status` function performs the check without including the contract size, users may be led to believe that a bid of 0 will be sufficient to cache their contract. This also conflicts with what is reported by `suggest_bid`.

Consider including the contract size in the capacity check or simply removing this part entirely in favor of `suggest_bid` to avoid confusion and potentially insufficient bids being placed.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Stylus-tools is at its first release stage. We are taking all input into consideration to improve the APIs for future releases.

L-09 Unbounded Ancestor Search For `rust-toolchain.toml` Can Lead to Incorrect Toolchain Usage

The `find_toolchain_file` walks parent directories to locate `rust-toolchain.toml` without any boundary checking, continuing until either finding the file or reaching the filesystem root. This unbounded search allows toolchain configuration from arbitrary ancestor directories, including system-wide locations or directories outside the project workspace. This can cause packages to inadvertently inherit toolchain settings from parent workspaces or sibling projects rather than using their intended configuration. In particular, this can affect reproducible builds, as the function may use a different `rust-toolchain.toml` than the project developer intended, leading to builds with incorrect Rust versions and non-reproducible artifacts.

Consider bounding the search to stop at the workspace root (obtainable from `cargo_metadata`) or, at minimum, the repository root. Doing so will prevent toolchain inheritance from outside the project's control.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> The search for rust-toolchain.toml is intended to mimic the rest of the Rust tooling.

L-10 Unpinned Docker Base Image Tag Undermines Reproducibility

The reproducible build workflow generates a Dockerfile that references the offchainlabs/cargo-stylus-base image using a `cargo_stylus_version` version tag instead of an immutable content digest (`@sha256:...`). Since Docker image tags can be re-targeted to different image contents at any time, this approach allows the referenced base image to change independently of the codebase.

If the base image tag is updated or maliciously re-targeted, subsequent reproducible builds may pull a different, potentially untrusted base image, breaking build determinism and undermining the reproducibility guarantee. It also introduces a potential supply-chain risk, as a compromised base image could execute arbitrary code within the container, which has write access to the user's project directory via a bind mount.

Consider pinning the base image to a specific content digest (`FROM repo@sha256:...`) or verifying the pulled image's digest against a known trusted value before building and running. If dynamic tag resolution is required, consider resolving the tag to a digest once via a trusted channel and embedding that digest into the generated Dockerfile.

In addition, consider limiting runtime privileges (for example, disabling host networking or using read-only mounts) to reduce potential impact if the base image is compromised. If pinning or verification cannot be implemented, consider clearly documenting this behavior to inform users of the associated reproducibility and security implications.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Stylus-tools is at its first release stage. We are taking all input into consideration to improve the APIs for future releases.

L-11 Vulnerable Dependencies

Running cargo audit revealed two known vulnerabilities, alloy-dyn-abi 1.3.1 and tokio-tar 0.3.1, in the transitive dependencies used by `stylus-tools`.

Consider upgrading them to their latest, non-vulnerable versions. Alternatively, consider refactoring or removing the affected code to eliminate reliance on vulnerable crates.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> We will be assessing our dependencies, and bumping the version of several crates (including these) before release.

L-12 Unchecked Type During ABI Decoding

Throughout the codebase, multiple instances of the `SolType::abi_decode_params` function being used to decode parameters were identified:

- Lines 138, 355, 369, and 383 in `stylus-proc::macros::sol_interace`
- Lines 571 and 680 in `stylus-proc::macros::public::types`
- Line 212 in `stylus_sdk::abi`

Consider using the more secure `SolType::abi_decode_params_validate` function to ensure correct decoding.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Stylus-tools is at its first release stage. We are taking all input into consideration to improve the APIs for future releases.

L-13 Unsanitized Function Identifiers Can Break Interoperability

The Stylus framework includes mechanisms for exporting a contract's ABI. This is handled by the Stylus SDK, which generates a Solidity-compatible output that can be imported into Solidity to interact with the Stylus contract. Therefore, the exported ABI must adhere to Solidity rules to guarantee compatibility. The `is_sol_keyword` function in `stylus_core::sol` helps detect such keywords, allowing to sanitize or to reject them. For example, Solidity keywords used as names in function parameters are prepended with an underscore using the `underscore_if_sol` function. However, function names face no restrictions. For instance, function names can be named after Solidity keywords. This is problematic when importing the export ABI to a Solidity file, preventing the contract from compiling.

Consider rejecting function names that match any Solidity keyword to avoid breaking interoperability.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Usage of these patterns is not supported. We will be updating our documentation to reflect this, and implemented proper error reporting at a later date.

L-14 Architecture Dependent Key Encoding Can Cause Discrepancies

In the `stylus_sdk::storage` module, the macro `impl_key` includes `usize` and `isize` as valid keys for mappings. Since `usize` and `isize` are pointer-sized, the representable range of these keys can differ across targets. For instance, tests commonly run natively, possibly on 64-bit architecture, while deployment targets the 32-bit `wasm32-unknown-unknown` architecture.

Using `usize` or `isize` as mapping keys can therefore derive different slots for the same key between local tests and on-chain execution, especially for values outside the 32-bit range. This risks silent state divergence, mismatched reads, or collisions when large keys are truncated on `wasm32`.

Consider disallowing target-dependent types as storage keys, providing a robust and deterministic code execution across targets.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> We have currently not encountered any issues with the current types, but we have made a note to ensure consistent cross-platform behavior in the future.

L-15 Non-Literal Fixed-Size Arrays Can Cause Selector Mismatches

Within the `stylus_proc::types` module, the `sol_interface` maps `syn` Solidity types to Alloy `SolType` using `SolidityTypeInfo`. For arrays, the code branches on `TypeArray::size()`. If `size()` is `Some`, a `FixedArray` is generated. Otherwise, a dynamic array is returned. Non-literal constant expressions such as `1+2` are not recognized by `size()`, causing fixed arrays to be treated as dynamic.

This results in both ABI and selector mismatches for declarations using constant-expression sizes. A declaration like `uint256[1+2]` is rendered as `uint256[]` in the selector preimage and as a dynamic array in the generated client, diverging from Solidity's canonical type `uint256[3]`. Calls using the generated client could revert due to selector mismatches or target a different overload.

Consider evaluating constant expressions for fixed array sizes in `sol_interface` before emitting types and selector strings, or rejecting non-literal sizes with a clear error.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Usage of these patterns is not currently supported. We will be updating our documentation to reflect this, and implemented proper error reporting or constant evaluation (if possible) at a later date.

L-16 Raw Actions Default to Copying Unbounded Return Data

The Stylus SDK offers mechanisms for performing raw calls and deployments within its `RawCall` and `RawDeploy` modules. These modules offer a powerful API for developers to initiate calls or deployments with an interface to granularly configure the action.

For instance, a caller can define the salt and the cache policy for a `RawDeploy` action, as well as call kind, value, gas, return data offset and size, and cache policy for a `RawCall` action. In the case of `RawDelpoy`, the `deploy` function will either return the deployed contract's address or attempt to copy the full return data if the deployment fails. Similarly, in `RawCall`, the `call` function will copy the full return data by default. For example, a malicious callee can return a very large buffer, forcing the caller to copy the entire return data and allocate enough memory to decode it, which can trigger linear memory growth charges or out-of-memory traps.

Although the return data size to copy is configurable in `RawCall`, it is not the case for `RawDeploy`. In both cases, however, the default behavior is to copy the full data, even though the user does not intend to do so, adding unnecessary gas costs to the operation. Furthermore, the generated client methods by the `sol_interface` macro rely on the safe wrappers around the `RawCall` module, performing a call, and decoding the returned bytes. However, users might not always want to copy the full return data or decode the returned data. In such a case, users must reimplement the calls to external contract functions to avoid extra costs.

Consider defaulting the size of the data to copy to 0, making the user opt-in to copy the return data when there is a need to decode the return parameters or propagate errors. Alternatively, consider adding a configuration to `RawDeploy` to limit or skip return data, similar to its `RawCall` counterpart configuration. In addition, consider enabling granular control over return data handling for generated client methods.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> We have not currently requested any additional features to the "raw" level API. If users want more control, they can use hostios directly to avoid uneccisary copies or other operations.

L-17 Replay Command is Broken on Windows

The replay command's implementation for spawning a debugger process differs between Unix and Windows. However, it is non-functional on Windows. On Unix systems, the code uses `cmd.exec()`, so control only returns and hits the `bail!` if `exec` fails. However, on Windows, the code uses `cmd.status()`, which always returns after the debugger exits. Therefore, the subsequent `bail!("failed to exec …")` runs on every successful replay and terminates cargo-stylus with an error.

Consider updating the Windows path to exit with the debugger's status instead of bailing.

Update: *Acknowledged, not resolved.*

Notes & Additional Information

N-01 Typographical Errors

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

- In line 10 of `cargo_stylus.rs`, "deployement" should be "deployment"
- In line 4 of `activation.rs`, "acitvation" should be "activation"
- In line 11 of `suggest_bid.rs`, "bid for in the cache manager" should be "bid for the cache manager"
- In line 163 of `mod.rs`, "Divegence" should be "Divergence"

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

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Noted for fix in a future release.

N-02 Silent Overflow in Gas Cost Estimation Returns Zero

The `print_gas_estimate_function` uses `gas_price.checked_mul(gas.into()).unwrap_or_default()` to calculate transaction costs, which silently returns zero on arithmetic overflow instead of propagating an error. This causes the CLI to display "0 ETH" for deployment and activation cost estimates when overflow occurs, potentially misleading users into believing transactions are free when they may actually be prohibitively expensive. While unlikely under normal conditions given `u128` capacity, extreme gas prices during network stress could trigger this behavior.

Consider returning an error on overflow instead of defaulting to zero.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Stylus-tools is at its first release stage. We are taking all input into consideration to improve the APIs for future releases.

N-03 Incorrect Assignment of WASM Lengths in Verification Failure

When a contract verification fails, the WASM lengths from the contract creation calldata and locally built contract are reported in the error message. However, their assignment appears to be swapped: the `calldata` should be used for `tx_wasm_length` and `deployment_data` for `build_wasm_length`. This mismatch reduces the accuracy of the reported error and can cause confusion during debugging of the source of failure.

Consider correcting the aforementioned variable assignments. In addition, consider renaming the arguments to `verify_create_deployment`. Doing so will ensure that the argument names accurately correspond to the source of the supplied codes.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Stylus-tools is at its first release stage. We are taking all input into consideration to improve the APIs for future releases.

N-04 Inconsistent Logging Pattern

Throughout the codebase, multiple instances of inconsistent logging methods were identified. These include `println`, `debug`, `info`, and `greyln`.

To improve code clarity, debugging, and maintainability, consider adopting a consistent approach towards logging patterns.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Stylus-tools is at its first release stage. We are taking all input into consideration to improve the APIs for future releases.

N-05 Lack of Explicit Check

The `cache_manager_address` function in `cache.rs` queries the `ARB_WASM_CACHE` precompile to retrieve a list of cache manager addresses. However, it assumes that the list will always contain a single entry. It then pops and returns the last element as the cache manager address. While this aligns with the current Arbitrum network architecture, which supports only one cache manager, this assumption may not hold in the future. If multiple cache managers are introduced, the current logic could lead to incorrect address selection or unexpected behavior.

Consider including an explicit check ensuring that only one address is returned. In addition, clearly document this architectural assumption to justify why the last address in the list is treated as the valid one.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> The cache manager currently will only ever return one value at a time. If there is ever a time in the future where we change this functionality, we will require a cargo-stylus update in order to take advantage of it.

N-06 Formatting Issues

Throughout the codebase, multiple instances of formatting issues were identified:

- In `stylus_sdk::call`, the `static_call`, `delegatecall`, and `call` functions can be refactored to configure the call using `flush_storage_cache` or `clear_storage_cache` instead of calling `host.flush_cache` manually.

- When exporting ABIs, there is no empty line between two interfaces. Consider adding an empty line to improve readability when exported to Solidity.

Consider addressing the above-listed formatting issues to improve the clarity and maintainability of the codebase.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> Noted for fix in a future release.

N-07 Unnecessary Flags for Special Function Names

The Stylus SDK currently requires explicit attributes to designate special functions, such as `constructor` (or `stylus_constructor`), `fallback`, and `receive`. The compiler also enforces a naming convention, for example, requiring the `#[fallback]` attribute to be used exclusively on a function named `fallback`.

This creates a redundancy where the function's role is declared twice: once by the `#[fallback]` attribute and again by the `fn fallback` name. This adds unnecessary complexity to the SDK and contract code, as the function's name alone could be used to detect its special purpose. Moreover, the logic for the special functions attributes is asymmetric. As mentioned above, functions with a reserved name such as `fallback` are required to be flagged with the `#[fallback]` attribute. However, the `#[fallback]` attribute can be used on functions with any other name.

Consider refactoring the SDKs' logic, relying solely on name-based detection for special functions. This change would streamline the SDK, remove superfluous attributes, and simplify the contract writing experience for developers.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> We have opted to avoid strictly "name-based" detection of special methods as explicity is better than implicit for readability and obvious behavior. The checks for the names are simply to avoid writing of contracts which use these names outside of their intended purpose.

N-08 Inconsistent Function Availability

The `RawCall` and `RawDeploy` structs offer many functions to configure a call or a deployment. For instance, the `flush_storage_cache` and `clear_storage_cache` functions [1] allow for configuring the cache policy invoked during the call or deployment.

These functions are crucial in reentrant mode to properly handle the cache before invoking external code. Therefore, in `RawDeploy`, both functions only exist in reentrant mode. However, in `RawCall`, these functions are always available, allowing users to define a custom cache policy which can be unnecessarily invoked before calling an external contract, potentially increasing execution cost.

Consider aligning the availability of `RawCall` cache configurations to match the `RawDeploy` counterparts, avoiding unnecessary function calls.

Update: *Acknowledged, not resolved. The Offchain Labs team stated:*

> We have opted to avoid strictly "name-based" detection of special methods as explicity is better than implicit for readability and obvious behavior. The checks for the names are simply to avoid writing of contracts which use these names outside of their intended purpose.

N-09 Unused WASM Build

The `exec` function invokes the normal Stylus WASM build, but the returned artifact and path are not referenced afterward. The command then proceeds to call the `build_shared_library` function that builds a host-native shared library (`.so` or `.dylib`), and then loads `user_entrypoint` from that native library. No code reads the WASM after this point. As such, while the WASM build can be useful for failing early if the contract no longer compiles to WASM, functionally, it is not required for the debugger flow.

To optimize the `replay` command, consider removing the wasm building process.

Update: *Acknowledged, not resolved.*

N-10 Unused `stable_rust` Flag

The `Args` struct in `replay.rs` defines a `stable_rust` flag. However, this flag is never used in the `exec` function.

Consider removing any unused flags to improve the clarity and maintainability of the codebase.

Update: *Acknowledged, not resolved.*

Conclusion

The Stylus SDK, core, and procedural macro crates have been refactored to include all `Host` interactions behind a single VM. In addition, cross-contracts calls and deployments have been rebuilt around a more explicit `Call` builder, while ABI generation has been upgraded to rely on unified AbiType encoders aligning Rust and Solidity semantics.

Storage types now expose clearer borrowing behavior, and procedural macros have been updated accordingly to generate selectors, encoders, and entry points that operate consistently in the new `VM` model. `cargo-stylus` has been reorganized into a modular structure with enhanced workspace support, improved deploy flows, and reproducible Docker-based verification.

The audit identified a critical-severity issue alongside several high- and medium-severity issues. These issues pertained to the CLI tool as well as the SDK, like call-context gating, selector construction, and storage-layout interoperability. Apart from these, a number of low-severity findings were also reported that centered on developer ergonomics and predictable behavior across targets.

Numerous security issues were uncovered related to Stylus CLI tool, along with several incomplete features. As such, the Arbitrum team is strongly advised to consider conducting further audits on the CLI portion of the codebase once all features have been implemented and thoroughly reviewed.

Overall, the codebase shows structural improvements and clearer abstractions compared to prior versions. Continued periodic reviews are recommended as Stylus matures and these refactors propagate into downstream workflows.