Security Audit Report

StylusPort Internal Review

v1.0

October 22, 2025

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security GmbH has performed an internal security audit of StylusPort.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/oak-security/stylusport
Scope	The scope has been segmented into three sprints: 1. Directories `handbook` and `examples` at commit `06d5a9b6e2be2263813080f0dd34821253c5199d` 2. Additional chapters in the `handbook` and `examples` directories: a. Testing & Debugging chapter at commit `ce8f10e6d54f72f7a5685ed2e62991aa93e92ea9` b. Gas Optimization chapter at commit `464f967bbf70b93d99a59d4c1805d67cc2b78b85`

	c. Security Considerations chapter at commit `38e73c183a6a3317b221bedabb2ff4742aa963ae` 3. The MCP server at commit `638d6d1138d25789ae09d9c72d1a0438500bd9a3`
Fixes verified at commit	`ed5f507921db51c24847ca532f9416258844bf3c` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

StylusPort is a comprehensive framework and toolkit to streamline the migration from SVM and Solana programs to Arbitrum Stylus smart contracts.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending**, **Acknowledged**, **Partially Resolved**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Low	The codebase is presented as a didactic example.
Code readability and clarity	High	The codebase is presented as a didactic example and is thoroughly described and well commented.
Level of documentation	High	The handbook is highly descriptive and provides a detailed explanation of the relevant codebase.
Test coverage		Not applicable.

Summary of Findings

No	Description	Severity	Status
1	Unauthorized token withdrawal via arbitrary source parameter in `create`	Critical	Resolved
2	Missing authorization check in `take_ownership` allows unauthorized ownership transfer	Critical	Resolved
3	Incorrect PDA signer and token authority in `unstake` causes Denial of Service on staked funds	Critical	Resolved
4	Missing ownership and mint constraints on token accounts in `Unstake` context	Major	Resolved
5	Unsanitized package name enables code injection in generated files	Major	Resolved
6	Missing mutable constraint	Minor	Resolved
7	Unbounded `WorkQueue` growth enables memory exhaustion	Minor	Resolved
8	Unbounded memory allocation from unrestricted line reads in server loop	Minor	Acknowledged
9	Unsafe logging of untrusted client input enables log injection and flooding	Minor	Resolved
10	Missing address validation in constructors and configuration functions	Minor	Resolved
11	Unbounded loop may cause out-of-gas revert for large schedules	Minor	Resolved
12	Build process fails on Docker and macOS ARM	Minor	Resolved
13	Incompatible Makefile syntax with macOS	Minor	Resolved
14	Missing experimental feature flags in Nix command execution	Informational	Resolved
15	Interactive code execution in handbook fails	Informational	Resolved
16	Hardcoded number of workers is not configurable	Informational	Resolved
17	Typographical and formatting errors across handbook documentation	Informational	Resolved

18	Parse errors do not return error messages	Informational	Resolved
19	Incorrect event emission using a static identifier instead of the new owner address	Informational	Resolved
20	Protocol version validation does not prevent downgrade attacks	Informational	Acknowledged

Detailed Findings

1. Unauthorized token withdrawal via arbitrary source parameter in `create`

Severity: Critical

In `examples/case-studies/bonafida-token-vesting/src/lib.rs:90-110`, the `create` function accepts a `source` address parameter and then attempts to transfer the total vesting amount from that source into the contract.

However, there is no check that the caller is authorized to initiate a transfer from `source`. The function does not require `source` to be the same as the caller, nor does it require an explicit signed consent from `source`.

This allows any attacker to call `create` with an arbitrary `source` who previously granted an allowance to the vesting contract, thereby pulling tokens from `source` into a vesting schedule controlled by the attacker via the `owner` and `destination` parameters.

Recommendation

We recommend restricting fund transfers. If third-party `source`s must be supported, require explicit authorization via signed permits.

Status: Resolved

2. Missing authorization check in `take_ownership` allows unauthorized ownership transfer

Severity: Critical

In `examples/concepts/errors-events/stylus/src/lib.rs:39-53`, the `take_ownership` function updates the contract's `owner` field to the message sender without verifying whether the caller is authorized to perform this action.

As a result, any caller can invoke `take_ownership` and assign ownership to themselves, effectively compromising contract control.

Recommendation

We recommend enforcing an authorization check to ensure a secure ownership transfer flow.

Status: Resolved

3. Incorrect PDA signer and token authority in `unstake` causes Denial of Service on staked funds

Severity: Critical

In `examples/concepts/fungible-tokens/anchor/src/lib.rs:57-75`, the `unstake` function uses an incorrect token authority and invalid PDA signer seeds when performing the token transfer through `transfer_checked`.

As a result, the unstake operation fails verification and reverts, preventing users from withdrawing their staked tokens.

This effectively locks user funds in the staking account, creating a denial-of-service condition for all stakers.

Recommendation

We recommend correcting the authority and PDA signer configuration to match the staking program's intended ownership model.

Status: Resolved

4. Missing ownership and mint constraints on token accounts in `Unstake` context

Severity: Major

In `examples/concepts/fungible-tokens/anchor/src/lib.rs:126-140`, the `Unstake` context definition lacks sufficient constraints on token accounts, particularly for `unstake_to_account`.

The current implementation does not verify that the destination account belongs to the caller or that it corresponds to the same mint as the staked token.

This omission can result in users unintentionally transferring tokens to incorrect or third-party accounts, leading to potential fund loss or operational inconsistencies.

Recommendation

We recommend strengthening the `Unstake` account validation by explicitly constraining `unstake_to_account` to ensure it is owned by the transaction signer and associated with the expected mint.

Status: Resolved

5. Unsanitized package name enables code injection in generated files

Severity: Major

In `mcp/src/tools.rs:178`, the `generate_stylus_contract_cargo_manifest` function uses the `package_name` parameter directly in a string template without sanitization. Similarly, the `generate_stylus_contract_main_rs` function only performs minimal sanitization with `.replace("-", "_")`. This allows injection of newline characters and arbitrary content into both the generated `Cargo.toml` and Rust source files.

An attacker could provide a malicious package name such as `mypackage"\n[dependencies]\nmalicious-crate = "1.0"` to inject arbitrary dependencies into the TOML manifest, or `mypackage\n malicious_code();\n //` to inject arbitrary Rust code.

While the server is local and developers typically create packages for themselves, the attack surface becomes exploitable if package names originate from external sources, or if a future development allows remote access. The injected malicious code would be compiled and potentially executed on the developer's machine, leading to system compromise.

Recommendation

We recommend sanitizing the package name to allow only alphanumeric characters and common safe characters such as underscore and hyphen, rejecting any input containing newlines or other control characters.

Status: Resolved

6. Missing mutable constraint

Severity: Minor

In `examples/concepts/native-token-handling/anchor/src/lib.rs:42-46`, the `deposit_account` field in the accounts struct is not marked as mutable.

As a result, clients generated from the Anchor IDL treat this account as read-only.

This oversight creates a usability and reliability issue, as users following the generated client code will consistently encounter failed withdrawals.

Recommendation

We recommend marking the `deposit_account` field as mutable by adding the `mut` constraint to its account attribute.

Status: Resolved

7. Unbounded `WorkQueue` growth enables memory exhaustion

Severity: Minor

In `mcp/src/server.rs:26-60`, the `WorkQueue` implementation stores incoming items in an internal `VecDeque` without any size limitation or backpressure mechanism.

The send method continuously pushes new items, and if consumers process them slower than producers add them, the queue grows indefinitely.

This allows a malicious or misbehaving client to flood the queue with requests, leading to uncontrolled memory consumption and eventual out-of-memory (OOM) termination of the server process.

Recommendation

We recommend implementing backpressure and queue size limits to prevent unbounded growth.

Status: Resolved

8. Unbounded memory allocation from unrestricted line reads in server loop

Severity: Minor

In `mcp/src/server.rs:104-113`, the server continuously reads input lines using `reader.read_line(&mut line)` without enforcing any maximum size.

The same `String` buffer is reused across iterations, allowing a malicious client to send an extremely large line or omit a newline entirely.

This behavior causes the server to allocate memory without bounds, leading to excessive memory consumption or potential denial of service (DoS) through resource exhaustion.

Recommendation

We recommend enforcing strict input size limits and read timeouts to prevent unbounded memory growth.

Status: Acknowledged

9. Unsafe logging of untrusted client input enables log injection and flooding

Severity: Minor

In `mcp/src/main.rs:17-44`, the functions `parse_client_msg` and `handle_client_msg` directly log untrusted client input and message content using eprintln!.

Because the data is written to logs without sanitization or truncation, a malicious client could inject terminal control sequences, escape characters, or excessively large payloads to manipulate log output or exhaust disk space.

Recommendation

We recommend sanitizing and bounding all untrusted data before logging.

Status: Resolved

10. Missing address validation in constructors and configuration functions

Severity: Minor

Multiple constructors and configuration functions across the codebase accept address parameters without performing validation checks. Specifically:

- In `examples/concepts/access-control/stylus/src/lib.rs:42-46`, the `constructor` function sets `authority` and `publisher` directly without verifying that the provided addresses are non-zero.
- In `examples/concepts/access-control/stylus/src/lib.rs:48-58`, the `update_config` method allows updating the publisher to a zero address, which may disable core functionality or break authorization logic.
- In `examples/concepts/cpi-to-external-call/stylus/src/lib.rs:34-38`, the constructor accepts any address, including zero or externally owned accounts (EOAs), where only valid contract addresses should be allowed.

Lack of address validation can lead to configuration corruption, denial of service (by assigning zero or invalid addresses), or unintended loss of contract control.

Recommendation

We recommend enforcing strict validation for all address inputs in constructors and configuration update functions.

Status: Resolved

11. Unbounded loop may cause out-of-gas revert for large schedules

Severity: Minor

In `examples/case-studies/bonafida-token-vesting/src/lib.rs:194`, the contract iterates through all schedule items using an unbounded loop. For schedules with a large number of items, this iteration may exceed the block gas limit, causing the transaction to revert and preventing users from interacting with their vesting schedules.

Recommendation

We recommend implementing the ability to unlock a schedule partially in case it contains too many items.

Status: Resolved

12. Build process fails on Docker and macOS ARM

Severity: Minor

The build process defined in the `README.md:54-57` fails when executed in Docker and on macOS ARM environments.

Specifically, running the command `make build`, triggers a dependency resolution failure during the compilation of the `ring` crate, which depends on `cc` and `shlex`.

This indicates missing or improperly configured build dependencies in the containerized and ARM-based environments.

Recommendation

We recommend ensuring all system-level build dependencies are installed before running the `make build` command.

Status: Resolved

13. Incompatible Makefile syntax with macOS

Severity: Minor

The `Makefile:1-22` uses the GNU `make` directive `.RECIPEPREFIX` to redefine the recipe prefix character from the default `TAB` to `>`.

However, this syntax is not supported by the default BSD `make` used on macOS. As a result, running any make target (e.g., make serve-book) on macOS fails with the missing separator error.

The error occurs because BSD `make` interprets the prefixed lines as invalid syntax rather than command delimiters. This prevents macOS users from building or serving the project without installing an alternative build tool.

Recommendation

We recommend refactoring the Makefile for cross-platform compatibility.

Status: Resolved

14. Missing experimental feature flags in Nix command execution

Severity: Informational

The Docker execution command provided in the `README.md:30-33` uses the default Nix container to initialize a development environment.

However, the command fails due to missing experimental feature flags required by newer versions of Nix. When executed as-is, it results in the following error:

"Experimental Nix feature 'nix-command' is disabled; add '--extra-experimental-features nix-command' to enable it, experimental Nix feature 'flakes' is disabled; add '--extra-experimental-features flakes'."

This error occurs because `nix develop` requires explicit enabling of the `nix-command` and `flakes` features, which are disabled by default in standard Nix configurations.

Recommendation

We recommend updating the command to explicitly enable the necessary experimental features by adding the `--extra-experimental-features` flag.

The corrected command should be:

```
nix develop --extra-experimental-features nix-command
--extra-experimental-features flakes
```

Status: Resolved

15. Interactive code execution in handbook fails

Severity: Informational

The `handbook/book.toml` configuration introduces the base setup for the project's documentation.

However, the generated handbook includes interactive code blocks that display a play button on hover, allowing users to execute embedded scripts directly from the documentation interface.

These execution features are non-functional in the current setup, resulting in user confusion and broken interactivity when attempting to run examples.

Recommendation

We recommend disabling the interactive execution feature for code snippets.

Status: Resolved

16. Hardcoded number of workers is not configurable

Severity: Informational

In `mcp/src/server.rs:10`, the number of workers is hardcoded as a constant with a value of `4`. This prevents users from adjusting the worker count based on their system resources, workload requirements, or performance needs.

Recommendation

We recommend making the number of workers configurable through an environment variable (e.g., in `.env`) to allow users to adjust it according to their needs.

Status: Resolved

17. Typographical and formatting errors across handbook documentation

Severity: Informational

Multiple sections within the handbook/src directory contain typographical and formatting inconsistencies that reduce documentation clarity and technical accuracy. The following issues were identified:

- `handbook/src/errors-events.md:18-24`:
    - Typographical error, “as show” should be corrected to “as shown.”
- `handbook/src/errors-events.md:42-47`:
    - Typographical error, “error occured” should be corrected to “error occurred.”
- `handbook/src/state-storage.md:7-8`:
    - Typographical error, “care must be take” should be corrected to “care must be taken.”
- `handbook/src/access-control.md:426-430`:
    - Misleading comment, incorrectly describes SolidityError as a trait instead of a derive macro.
- `handbook/src/introduction.md:24-29`:
    - Markdown formatting issue, bullet points under Business benefits are rendered on a single line instead of separate list items.

Recommendation

We recommend reviewing and correcting all typographical, grammatical, and markdown formatting issues

Status: Resolved

18. Parse errors do not return error messages

Severity: Informational

In `mcp/src/main.rs:20`, the `parse_client_msg` function returns `None` when JSON deserialization fails. The error is only logged to `stderr`, which means the client does not receive any feedback about the parsing failure through the communication stream. If the server is exposed for remote access, this lack of error feedback may impair user interaction and make debugging difficult.

Recommendation

We recommend returning a JSON-formatted error message to the client for consistency with the response that is sent on success (`let Ok(client_msgs) = serde_json::from_str(input)`)

Status: Resolved

19. Incorrect event emission using a static identifier instead of the new owner address

Severity: Informational

In `examples/concepts/errors-events/anchor/src/lib.rs:25-33`, the `emit_event` function emits an `OwnerChanged` event where the `current_owner` field is incorrectly set to the static ID value rather than the actual new owner's public key.

This results in misleading event data, as the emitted event does not reflect the true ownership transition.

Recommendation

We recommend modifying the event emission to include both relevant addresses.

Status: Resolved

20. Protocol version validation does not prevent downgrade attacks

Severity: Informational

In `mcp/src/handler.rs:11`, the `initialize_request` function validates that the client's protocol version does not exceed `LATEST_PROTOCOL_VERSION`. However, the implementation appears to allow older protocol versions to maintain backward compatibility. While backward compatibility is beneficial, the system does not prevent clients from negotiating vulnerable or deprecated protocol versions if such versions exist.

This is a theoretical concern, but downgrade attacks are a known attack vector in web2 protocols where an attacker forces the use of an older, vulnerable protocol version. If any past protocol versions contain security vulnerabilities, malicious clients could exploit them by requesting those versions.

Recommendation

We recommend implementing a minimum supported protocol version check to ensure that deprecated or vulnerable protocol versions cannot be used.

Status: Acknowledged