Code Assessment
of the PAU assemblers Smart Contracts

PUBLIC

Date June 12, 2026

Produced for Sky

By CHAINSECURITY

Contents

1 Executive Summary

Dear all,

Thank you for trusting us to help Sky with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of PAU assemblers according to **Scope** to support you in forming an opinion on their security risks.

Sky implements `DefaultPAUAssembler`, a permissionless helper contract that deploys and fully configures a complete PAU stack together with any number of `AdministeredAgent` allocators. It assigns all administrative and operator roles to caller-supplied addresses and renounces every role it holds during setup, retaining no privilege over the deployed contracts.

The most critical subjects covered in our audit are access control and functional correctness. The general subjects covered are documentation and trustworthiness.

Minor inconsistencies between the documentation and the implementation were found and reported, see **Specification Mismatches and Inconsistencies**.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,
ChainSecurity

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the PAU assemblers repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

Version	Date	Commit Hash	Note
1	9 June 2026	9f3f606e1243b5b0403a1a7e541311bc05fe7150	Initial Review
2	12 June 2026	d7d6f084604d4f7e45879a3b15d03ee870231467	v1.0.0

For the Solidity smart contracts, the compiler version `0.8.34` was chosen.

The files in scope are:

```
src/
├── DefaultPAUAssembler.sol
└── interfaces/
    └── IDefaultPAUAssembler.sol
```

2.1.1 Excluded from scope

All other files, including the tests and scripts, are out of scope.

In particular, the underlying contracts the assembler interacts with are out of scope and are relied upon to behave as expected:

- PAU Administered Agent contracts, i.e. the `AdministeredAgent` and its `AdministeredAgentFactory`; and
- Diamond PAU contracts, i.e. the `PAUFactory` and the `AccessControls`, `ALMProxy`, `RateLimits`, and `Controller` it deploys.

3 System Overview

This system overview describes the initially received version (**Version 1**) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

`DefaultPAUAssembler` is a permissionless helper that deploys and wires a complete PAU instance, together with its allocator agents, in a single `deploy` call. It gives a deterministic, reviewable alternative to performing the deployment and role wiring by hand (the underlying stack is Diamond PAU).

Anyone may call `deploy`, and the flow has four parts:

1. It deploys the core contracts: one `AccessControls`, one `ALMProxy`, one `RateLimits`, and one `Controller`, plus a set of `AdministeredAgent` instances.
2. It configures each agent from the supplied config, assigning its sets of `admin`, `actor`, `grantor`, and `revoker` addresses.
3. It grants the system roles:
   - `AccessControls`, `ALMProxy`, and `RateLimits` each receive `DEFAULT_ADMIN_ROLE` for their own distinct set of supplied addresses.
   - Each deployed agent receives `ALLOCATOR_ROLE` on `AccessControls` so it is privileged to perform allocator actions on the PAU.
   - The `Controller` receives the `CONTROLLER` role on `ALMProxy` and `RateLimits` so it can operate them.
4. If integration IDs are supplied, the initial facet configuration is loaded through a call to `updateIntegrations`. An empty array skips this step, so a stack can be deployed without integrations and configured later.

The assembler holds admin privileges on the deployed contracts and agents only during setup, relinquishing them before returning so it has no power over the system once `deploy` completes.

4 Trust Model

The `DefaultPAUAssembler` defines no persistent privileged roles of its own: `deploy` is permissionless, and the assembler renounces every role it holds during setup before the call returns. Trust therefore concerns who calls `deploy`, the addresses a deployment grants authority to, and the external contracts the assembler relies on.

Caller of `deploy` (*untrusted*). Anyone may call `deploy`; it has no access control. The caller fully determines the resulting configuration (the `DEFAULT_ADMIN_ROLE` holders of the components and the admins/actors/grantors/revokers of each agent) but cannot affect any previously deployed stack.

Configured admins, actors, grantors, and revokers (*trust deferred*). The assembler only assigns these roles as instructed; the trust placed in them is that of the deployed PAU components and the `AdministeredAgent` (out of scope) to which the roles apply.

External factories (*fully trusted*). The `pauFactory` and `administeredAgentFactory` addresses are fixed at construction. They are trusted to deploy genuine PAU components and `AdministeredAgent` instances that behave as specified. It is further assumed that any `integrationIds` are pre-registered on the underlying PAU factory's beacon.

5 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- **Likelihood** represents the likelihood of a finding to be triggered or exploited in practice
- **Impact** specifies the technical and business-related consequences of a finding
- **Severity** is derived based on the likelihood and the impact

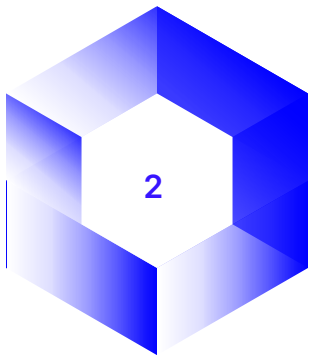
We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	**Impact**		
	High	Medium	Low
High	**Critical**	**High**	**Medium**
Medium	**High**	**Medium**	**Low**
Low	**Medium**	**Low**	**Low**

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

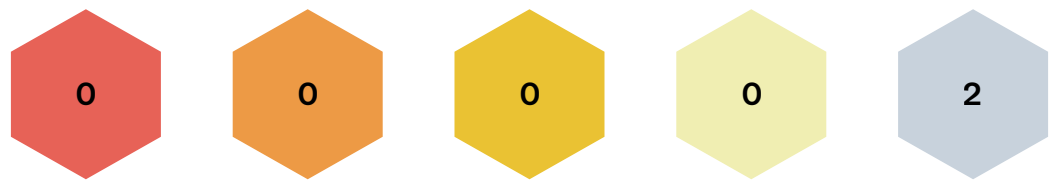
6 Findings

In this section, we describe the findings identified during the course of the engagement. The findings are categorized by severity as explained in the Terminology section. Below we provide an overview of the total number of findings per severity, as well as the number of open issues.



Open Issues

Total number of findings



Critical | High | Medium | Low | Informational

6.1 Overview

The following table lists all identified findings along with their severity and current status. A finding is considered resolved once it has been fully corrected or the specification has been changed accordingly. Findings with any other status, including partial fixes, acknowledgements, and accepted risks, remain open.

Info	#001	Effective `AdminConfig` Admins Can Be Fewer than Supplied	Acknowledged
Info	#002	Specification Mismatches and Inconsistencies	Partially Changed / Acknowledged

6.2 Descriptions

#001 Effective `AdminConfig` Admins Can Be Fewer than Supplied

Informational	Version 1	Acknowledged

The set of admins actually granted on a component can be smaller than the array passed in `accessControlAdmins`, `proxyAdmins`, or `rateLimitsAdmins`, with no error raised:

- Granting `DEFAULT_ADMIN_ROLE` is idempotent, so listing the same address twice grants it once.
- If a list includes the assembler itself, the grant is undone moments later when the assembler renounces `DEFAULT_ADMIN_ROLE` during teardown. Were it the only entry, the **component would be left with no admin at all**.

By contrast, the per-agent configuration arrays are unforgiving: a duplicate there reverts the entire deployment rather than collapsing silently.

Acknowledged

The behaviour is intentional and documented: the README and the deployment checklist state that duplicate entries in `adminConfig` arrays are idempotent no-ops and are not guarded on-chain. Callers are expected to supply distinct, non-assembler addresses.

Passing the assembler's own address in any admin array, which causes the self-granted role to be revoked during teardown and can leave a component with no admin, remains unaddressed in both code and documentation.

#002 Specification Mismatches and Inconsistencies

Informational	Version 1	Partially Changed / Acknowledged

Several places where the documentation and the implementation disagree:

1. The README and CHECKLIST reference errors that do not exist in the codebase, `NoAdmins` and `ZeroAdmin`, and collapse the agent/component distinction into one name. The actual errors are `NoAgentAdmins`, `NoDefaultAdmins`, and `ZeroDefaultAdmin`.
2. The `deploy()` natspec documents the `CONTROLLER` grant on the ALMProxy only, but the implementation also grants `CONTROLLER` on RateLimits. The README is consistent with the code here; the on-contract natspec is not.
3. The README and CHECKLIST state that one or more allocators are expected, but the implementation accepts an empty `allocatorAgentConfigs` and deploys a stack with no allocator.
4. The interface natspec describes the deployed stack as `(ALMProxy, RateLimits, Controller)`, omitting AccessControls, even though `deploy()` deploys it, wires every role on it, and returns it. The README defines the PAU stack as including AccessControls.

Partially Changed

Items 2 and 3 were corrected: the `deploy()` natspec now documents the `CONTROLLER` grant on `RateLimits`, and the README and CHECKLIST were updated to reflect that an empty `allocatorAgentConfigs` is accepted.

Items 1 and 4 were corrected only partially:

- Item 1: The stale error names were replaced in the README, but the CHECKLIST still attributes an empty per-agent `admins` array to `NoDefaultAdmins`. The actual error is `NoAgentAdmins`, so the agent/component distinction remains collapsed into one name there.
- Item 4: The `deploy()` natspec now lists AccessControls as part of the deployed stack, but the contract-level natspec of the interface still describes the stack as `(ALMProxy, RateLimits, Controller)`, omitting AccessControls.

Acknowledged

Sky deployed the contract already.

7 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.