Code Assessment
of the PAU Administered Agent Smart Contracts

PUBLIC

Date June 08, 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 Administered Agent according to **Scope** to support you in forming an opinion on their security risks.

Sky implements `AdministeredAgent`, a contract that holds a privileged role within the Diamond PAU (Spark Liquidity Layer) system, initially the allocator role, and routes operator calls into the PAU on its behalf.

The most critical subjects covered in our audit are access control and functional correctness. The general subjects covered are documentation and trustworthiness. Security regarding all the aforementioned subjects is good.

In summary, we find that the codebase provides a good 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 Administered Agent 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	2 June 2026	5c267327a1abda869a5d3a76706a3de26b0d8da2	v1.0.0-beta.0
2	8 June 2026	bfaaf709a8664d74d12604455f0365a0a12439cf	v1.0.0

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

List the files in scope using a filetree block. Use 4-space indentation for nesting. The renderer auto-generates tree-drawing characters.

```
src/
├── AdministeredAgent.sol
├── AdministeredAgentFactory.sol
└── interfaces/
    ├── IAdministeredAgent.sol
    └── IAdministeredAgentFactory.sol
```

2.1.1 Excluded from scope

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

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.

The `AdministeredAgent` is a contract intended to be granted a privileged role within the Diamond PAU (Spark Liquidity Layer) system, initially the allocator role, so that it can route operator calls into the PAU on its behalf. It is intended to replace the `ALMProxyFreezable` and to act as an access-control wrapper in front of the Diamond PAU controllers.

The contract manages four independent sets of addresses:

- **Admins** govern role configuration: they add and remove admins, grantors, and revokers, and may also add and remove actors. The contract enforces that at least one admin always remains.
- **Grantors** may add actors.
- **Revokers** may remove actors.
- **Actors** may execute arbitrary calls through the agent.

Splitting the right to add an actor (`grantors`) from the right to remove one (`revokers`) is intended so that onboarding and emergency offboarding can be controlled by separate authorities with independent multisig thresholds.

Actors execute operations through three functions: `call`, `batchCall` and `sendValue`. Each forwards to the given target(s), so the agent acts with the authority of whatever PAU role it holds.

The `AdministeredAgentFactory` is a minimal deployer that instantiates a new `AdministeredAgent` with a single initial admin.

4 Trust Model

The agent holds a privileged role within the Diamond PAU (initially the allocator role), so any address that can execute through it inherits that authority.

Admins (*fully trusted*). Govern the full role configuration (admins, grantors, revokers) and may also add and remove actors directly, making a single admin a superset of the grantor and revoker authorities. The only enforced constraint is that at least one admin must remain. Expected to be governance.

Grantors (*partially trusted*). May add actors, intended for higher-threshold onboarding of a replacement actor during incident response. Since a grantor may add itself, it can effectively obtain actor authority and is trusted not to.

Revokers (*partially trusted*). May remove actors, intended as a lower-threshold watchdog for rapid offboarding. A malicious revoker can only remove legitimate actors and halt operations. Removal prevents future calls but cannot reverse or interrupt actions an actor takes before (or in front of) the removal.

Actors (*partially trusted*). May execute arbitrary calls (`call`, `batchCall`, `sendValue`), wielding the agent's full PAU authority and any ETH it holds. Because the target and calldata are arbitrary, this also extends to any ERC20 or other contract the agent can interact with (transfers, approvals, and similar), though the agent is not expected to hold token balances or be privileged otherwise in an unexpected way. Actors are the allocator operators (multisigs or EOAs), bounded only by the downstream PAU controllers, not by the agent.

Unprivileged callers (*untrusted*). May fund the agent via `receive`, read getters, and deploy new agents via the factory. None of this grants authority over an existing agent; integrators must verify a factory-deployed agent's admin, not just its provenance.

5 System Considerations

We leverage this section to highlight findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require a modification inside the project. Instead, they should raise awareness in order to improve the overall understanding for users and developers.

SC1 Value Handling in Actor Calls

System Consideration	Version 1

The three actor functions handle ETH inconsistently, both in whether they may spend the agent's held balance and in how they treat excess `msg.value`.

1. `call` forwards exactly `msg.value` to the target and never draws on the agent's held balance, making it the only actor function that cannot spend the agent's own funds.
2. `batchCall` with `sum(values) < msg.value` leaves the surplus in the agent rather than refunding `msg.sender`.
3. `batchCall` with `sum(values) > msg.value` covers the difference from the agent's held balance.
4. `sendValue` sends `value` from the agent's balance regardless of `msg.value`; any `msg.value` supplied is left in the agent, and the local balance is spent.

6 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

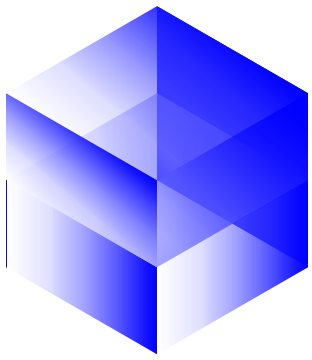
We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

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.

7 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.



No Open Issues

Total number of findings

Critical	High	Medium	Low	Informational
0	0	0	0	1

7.1 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	The Agent Can Be Granted a Role over Itself	Code Corrected

7.2 Descriptions

#001 The Agent Can Be Granted a Role over Itself

Informational	Version 1	Code Corrected

The role-granting functions only reject the zero address, so `address(this)` can be added to any role set. Since actors execute arbitrary calls through the agent, a self-call's `msg.sender` is `address(this)`. If `address(this)` were granted a role, the actors would thus automatically inherit that role:

- **Admin**: an actor self-calls `addAdmin` and gains full admin authority.
- **Grantor**: an actor self-calls `addActor` and gains grantor authority.
- **Revoker**: an actor self-calls `removeActor` and gains revoker authority.

This could allow actors to escalate their privileges but requires an admin to misconfigure the roles.

> **Code Corrected**
>
> The code has been adjusted. More specifically, roles can be still self-granted. However, no self-calls can be executed anymore.

8 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.