Code Assessment
of the Diamond PAU v1.14 Smart Contracts

PUBLIC

Date July 06, 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 Diamond PAU v1.14 according to **Scope** to support you in forming an opinion on their security risks.

Sky implements two new facets integrating NFAT facilities into the PAU system. An NFAT facility is a lending venue: `NFATPrimeFacet` implements the lender-side flows (subscribing capital, withdrawing it, and collecting repayments), while `NFATHaloFacet` implements the borrower-side flows, drawing principal against issued NFATs and servicing it while tracking outstanding principal and linearly accruing interest per position.

The most critical subjects covered in our audit are functional correctness, security of assets and access control. The per position accounting tracks outstanding principal and non-compounding interest and bounds repayments accordingly. It relies on facility configuration that is not enforced on-chain, see **Issue Accounting and Rate Limit Are Not Bound to the Facility Gem**. Security regarding all the aforementioned subjects is high.

The general subjects covered are documentation, trustworthiness, and event handling. The codebase features extensive NatSpec and a dedicated integration document; we reported several mismatches between documentation and implementation, see **NatSpec and Documentation Mismatches**. The trust model's containment of a compromised allocator holds for asset flows; for data-only actions and the facility configuration the facets rely on off-chain, see **Data-Only Subscribe Is Not Restricted for a Compromised Allocator** and **Issue Does Not Enforce That the Facility Recipient Is the ALMProxy**. Security regarding all the aforementioned subjects is good.

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 Diamond PAU v1.14 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	4812e320fef3ad6eb71de218df6c3b1f486139fd	v1.14.0-beta.0
2	24 June 2026	9d4c3a896ea187b98ff53ce837e239504eb75257	Fixes
3	6 July 2026	cbf71b2ac840ca9288eb867d3dd354e08089e1d7	v1.14.0

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

```
src/
├── facets/
│   ├── nfat-halo/
│   │   ├── INFATHaloFacet.sol
│   │   └── NFATHaloFacet.sol
│   └── nfat-prime/
│       ├── INFATPrimeFacet.sol
│       └── NFATPrimeFacet.sol
└── libraries/
    └── RateLimitHelpers.sol
```

2.1.1 Excluded from scope

All other contracts in the repository as well as the NFAT Facility. However, these contracts have been covered in other reviews.

Additionally, all dependencies, scripts test and similar are out of scope.

3 System Overview

This system overview describes the changes introduced in version **Version 1** of the contracts as defined in the Assessment Overview. This review is scoped exclusively to the NFAT integration added in this version: the two new facets `NFATPrimeFacet` and `NFATHaloFacet`.

3.1 NFAT Integration

An **NFAT facility** (`NFATFacility`) is a lending venue where capital is subscribed (gem token, e.g. USDS) and turned into a non-fungible position (an NFAT NFT) that represents a specific tranche of deployed principal. Repayments of principal and interest accrue against the NFT's `tokenId` and are later collected by the NFT owner.

The NFAT design involves the two opposite sides (lender and borrower):

- **Prime** — the lenders. They deposit gem into the facility, receive NFAT NFTs on issuance, and collect repayments once the borrower services the debt.
- **Halo** — the borrowers. They draw principal from the facility by triggering issuance and service the resulting debt via repayments.

This split maps onto two dedicated facets described below.

3.1.1 NFATPrimeFacet

`NFATPrimeFacet` implements the lender-side flows. It has no admin configuration. All three functions are gated on `ALLOCATOR_ROLE` and measure their rate-limit decrement from the **actual ALMProxy gem balance delta**.

1. **`subscribe(facility, amount, data)`**: approves `gem` to the facility, calls `facility.subscribe(amount, data)`, and resets the approval. The gem actually spent (balance delta) decrements the rate limit. `amount == 0` is supported: the call still forwards `data` to the facility and emits an event, but no gem moves and the rate limit is unchanged.
2. **`withdraw(facility, amount)`**: calls `facility.withdraw(amount)` to reclaim unissued subscribed capital back to the ALMProxy. The gem actually received (balance delta) decrements rate limit.
3. **`collect(facility, tokenId, amount)`**: calls `facility.collect(tokenId, amount)` to pull repaid capital back to the ALMProxy. The gem actually received decrements the rate limit.

Rate-limit key summary:

Operation	Key tuple	Rate-limit constant
`subscribe`	`(gem, facility)`	`LIMIT_NFAT_PRIME_SUBSCRIBE`
`withdraw`	`(facility)`	`LIMIT_NFAT_PRIME_WITHDRAW`
`collect`	`(facility)`	`LIMIT_NFAT_PRIME_COLLECT`

3.1.2 NFATHaloFacet

`NFATHaloFacet` implements the borrower-side flows. It introduces per-facility state (a cumulative interest index) and per-position bookkeeping to track outstanding principal and accrued interest for each `tokenId`.

The facet maintains a facility-wide cumulative interest index that advances linearly over time:

```
index(now) = index(lastCheckpoint) + annualGrowthRate × (now − lastCheckpoint) / 365 days
```

Each position snapshots the facility index at issue time. When a position is next touched, the index delta is applied to `outstandingPrincipal` to materialise newly accrued interest:

```
outstandingInterest += outstandingPrincipal × (currentIndex − position.interestIndex) / 1e18
position.interestIndex = currentIndex
```

The following functions are gated on `DEFAULT_ADMIN_ROLE`:

1. **`setAnnualGrowthRate(facility, annualGrowthRate)`**: configures the per-facility annual growth rate used for interest accrual. `annualGrowthRate` is a 1e18-scaled APR (`1e18 == 100%/year`). The call first checkpoints the facility so interest accrued under the previous rate is preserved.

The below functions are gated on `ALLOCATOR_ROLE` and nonreentrant.

1. **`issue(facility, to, tokenId, amount)`**: checkpoints the facility, verifies the position does not yet exist, and calls `facility.issue(to, tokenId, amount)` via the ALMProxy. The facility mints the NFAT to `to` and transfers drawn gem to its configured `recipient` (the ALMProxy). The gem received (balance delta) is recorded as `Position.outstandingPrincipal` and decrements the rate limit. Issuing with `amount == 0` reverts.

   The issue rate limit is keyed on `to` (the NFAT recipient) rather than on a `tokenId`. This doubles as a whitelist of inbound actors: only recipients with a configured, non-zero limit can be issued to, keeping drawn principal inside the Sky ecosystem. It also avoids a governance bottleneck where a limit would have to be pre-provisioned for every individual `tokenId`.

2. **`repayPrincipal(facility, tokenId, amount)`**: checkpoints the position (accruing outstanding interest first), enforces `amount <= outstandingPrincipal`, approves gem, calls `facility.repay(tokenId, amount)`, resets the approval, and measures the gem actually spent (balance delta). Decrements `Position.outstandingPrincipal` and the rate limit.

3. **`repayInterest(facility, tokenId, amount)`**: identical flow to `repayPrincipal` but enforces `amount <= outstandingInterest` and decrements `Position.outstandingInterest` and the rate limit.

Principal and interest are serviced through separate entry points with separate rate limits, keeping the two obligations independently throttleable. Interest continues to be repayable even after principal is fully retired.

Rate-limit key summary:

Operation	Key tuple	Rate-limit constant
`issue`	`(facility, to)`	`LIMIT_NFAT_HALO_ISSUE`
`repayPrincipal`	`(gem, facility)`	`LIMIT_NFAT_HALO_REPAY_PRINCIPAL`
`repayInterest`	`(gem, facility)`	`LIMIT_NFAT_HALO_REPAY_INTEREST`

4 Trust Model

This trust model covers only the components introduced by the NFAT integration: the `NFATPrimeFacet` and `NFATHaloFacet` and their interaction with the external NFAT facility. The trust model of the underlying PAU framework (Controller, ALMProxy, RateLimits, facet wiring) is described in the full Diamond PAU report.

4.1 Roles

Admin (`DEFAULT_ADMIN_ROLE`): Fully trusted. Within the new facets, the admin's only function is `setAnnualGrowthRate` on the `NFATHaloFacet`, which governs how quickly interest accrues on a facility's issued positions. A wrong rate misstates the interest bookkeeping but does not by itself move funds. (At the PAU level, the admin is fully trusted regardless, as it wires facets and configures rate limits.)

Allocator (`ALLOCATOR_ROLE`): Semi-trusted. Within the configured rate limits, the allocator controls the timing and sizing of all capital flows between the ALMProxy and the facility. It cannot direct funds to arbitrary addresses: gem only ever moves between the ALMProxy and the facility, and the `issue` rate limit is keyed per `(facility, to)`, acting as a whitelist of NFAT recipients. The worst an allocator can do is griefing via mistimed or unwanted (but whitelisted) flows.

4.2 External Components

NFAT facility (`NFATFacility`): An immutable, non-upgradeable contract with an immutable `gem`. Its own privileged roles become part of the trust model:

- **Facility wards** (`auth`): Must be fully trusted, they can steal funds. A ward can drain all facility-held gem (`rescue`, `rescueDeposit`, `rescueCollectable`) and redirect issuance proceeds by changing the `recipient` address. Wards also manage all other facility roles and the identity network.
- **Operator** (`toll`): Only gates `issue`. This role must be held by the ALMProxy so the `NFATHaloFacet.issue` flow can execute; it grants exactly the rights needed for that one facet and nothing more. Any additional operator could pre-empt issuance of a given `tokenId`.
- **Freezer** (`cop`): Can `stop` the facility, blocking `subscribe`, `issue`, `repay` and `collect` (`withdraw` stays live). Denial of service only; only a ward can `start` again.

Identity network: An external membership registry configured by the facility wards. It gates NFAT minting/transfers and `collect`. It is trusted to report membership correctly and to remain available: it cannot move funds, but a malicious or stale registry blocks issuance and collection.

Gem: Assumed to be a well-behaved ERC-20 (e.g. USDS). The facets measure every flow as the actual ALMProxy balance delta rather than trusting the requested amount, so partial fills are accounted correctly; the position and interest bookkeeping nevertheless assumes the gem does not rebase.

4.3 Operational Preconditions

- On the Halo side, the facility's `recipient` must be the ALMProxy and the ALMProxy must hold the operator role. If `recipient` points elsewhere, `issue` mints the NFAT and pays out the principal to that other address while recording `outstandingPrincipal = 0`.
- On the Prime side, the ALMProxy must be a member of the identity network (if one is set); otherwise it can neither receive NFATs at issuance nor `collect` repayments.

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 Data-Only Subscribe Is Not Restricted for a Compromised Allocator

System Consideration | Version 1

The trust model assumes a compromised allocator is contained by the rate limits. For asset movements this holds: flows are bounded by the configured limits and settle between the ALMProxy and the facility.

`NFATPrimeFacet.subscribe()` additionally forwards an arbitrary `data` payload to `NFATFacility.subscribe()`, which is intended for off-chain agreements. This action is not restricted: a compromised allocator can emit arbitrary agreement `data` from the ALMProxy identity, in unlimited frequency, for any facility with a configured subscribe limit. Note that publishing also accepts `amount == 0` (data-only calls, as supported by the facility) which allows the agreement updates even when no tokens are available.

SC2 Subscribe Rate Limit Is Not Keyed on the Facility Recipient

System Consideration | Version 1

`getSubscribeRateLimitKey` does not key on the facility's recipient (the Halo side). The recipient can change in the facility.

Keying the limit on the recipient would enforce that, at the moment funds are pushed from the ALMProxy to the facility, the recipient is the expected one. Note the recipient can change at any time afterwards, so subscribed capital could still be drawn to a changed recipient.

SC3 Issue Does Not Enforce That the Facility Recipient Is the ALMProxy

System Consideration | Version 1

`issue` does not assert that the facility's recipient is the ALMProxy but sizes the recorded principal and the consumed issue rate limit from the ALMProxy gem balance delta. The recipient being the ALMProxy is an off-chain expectation, not enforced on-chain.

If the recipient is not the ALMProxy, the principal is paid out to that other address and the proxy balance delta is zero. As a result:

- The issue rate limit is decreased by that zero delta, so it is not consumed and offers no bound on this flow.
- The position records `outstandingPrincipal = 0`, so no claim is created on the Halo side and nothing later treats the proxy as owing those funds; the accounting does not go out of step.

The facet relies on the off-chain expectation that the facility recipient is the ALMProxy.

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

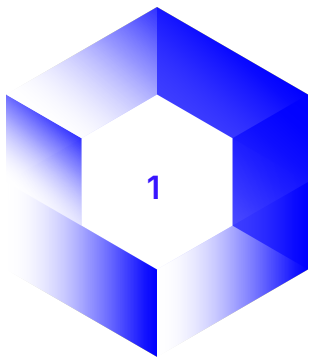
We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

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.

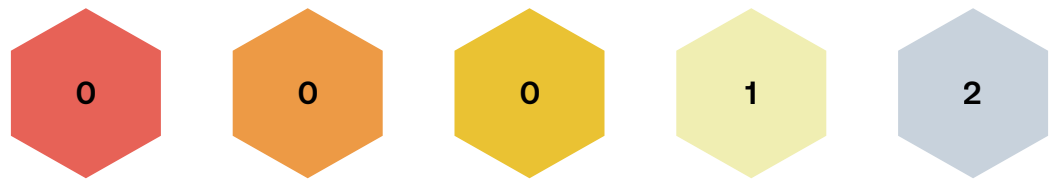
7 Findings

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.



Open Issues

Total number of findings



Critical High Medium Low Informational

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. Non-informational findings with any other status, including partial fixes, acknowledgements, and accepted risks, remain open.

Low	#001	Issue Accounting and Rate Limit Are Not Bound to the Facility Gem	Risk Accepted
Info	#002	Growth Rate Set After Issuance Permanently Lowers the Interest Repayment Cap	Acknowledged
Info	#003	NatSpec and Documentation Mismatches	Spec. Part. Changed / Acknowledged

7.2 Descriptions

#001 Issue Accounting and Rate Limit Are Not Bound to the Facility Gem

Security	Low	Version 1	Risk Accepted

The issue rate limit is keyed on `(facility, to)` with no gem, and `outstandingPrincipal` is stored as a bare token-unit count with no record of which gem it was denominated in. Repay, in contrast, keys its limits on `(facility, gem)`, so the two sides disagree on whether the gem is pinned. If a facility's gem ever changed between issue and repay, a position opened in one token could be repaid in another, and the same unit count would represent a very different value:

1. `gem` is USDC, issue and repay limits configured for USDC. Assume they correspond to 1M USDC (`1e12`).
2. `gem` changes to USDS.
3. `issue` runs: 1e12 USDS wei (worth 1e-6 USDS) is received and `1e12` is recorded as `outstandingPrincipal` and consumed for the issue rate limit.
4. `gem` changes back to USDC.
5. `repay` runs against the recorded principal: `1e12` is now 1M USDC, so the ALMProxy pays out ~1M USDC to settle a position worth almost nothing.

The flow drains funds, and the issue rate limit never bounded the real value at risk. The underlying problem is that the promise of a future repay (`issue`) is not strict enough about the expected parameters (i.e. `gem`).

The NFAT facility currently has an immutable gem, so this is not reachable as written; the exposure is the missing gem binding on the issue side and on the stored principal, which leaves the accounting trusting that the gem never changes.

> **Risk Accepted**
>
> Sky's rate-limit key derivation standard includes only the addresses a facet interacts with, not those it merely reads from. In `issue`, the gem is only read to query the balance delta, so it is not part of the key. With the legitimate facility, where the gem is immutable, a gem differing between issue and repay is not possible.
>
> Should a misbehaving facility with a mutable gem be used, there is no protection: a later repayment is capped by the principal recorded under the old gem and can move funds out of the ALMProxy against an incorrect cap.

#002 Growth Rate Set After Issuance Permanently Lowers the Interest Repayment Cap

Informational	Version 1	Acknowledged

`NFATHaloFacet` tracks `outstandingInterest` per position as the upper bound for `repayInterest()`. The accrual rate `annualGrowthRate` defaults to zero and is configured independently of the rate limits that gate `issue()`, so a position can be issued before the rate is set and is then tracked as interest-free.

Setting the rate later does not backfill: `setAnnualGrowthRate()` checkpoints the facility at the old (zero) rate before storing the new one. No interest is ever accounted for the period before the rate was set, and the cap enforced by `repayInterest()` is lowered accordingly. In general, rate changes only apply to time elapsed after the change.

Acknowledged

Sky notes that a growth rate of zero is a valid interest rate and that issuance is intentionally gated by the rate limits alone, not by the growth rate being set.

#003 NatSpec and Documentation Mismatches

Informational	Version 1	Specification Partially Changed / Acknowledged

Below is a non-exhaustive list of mismatches between the NatSpec or documentation and the implementation. None affect on-chain behavior.

1. The NatSpec of `getFacilityState()` in `INFATHaloFacet` states that the returned index includes time-based accrual since the last checkpoint. The implementation returns the stored values `state.interestIndex` and `state.lastUpdated` without applying accrual for the time elapsed since `lastUpdated`. The live index is computed by `_getCurrentInterestIndex()`, which is not used here.

2. The NatSpec of `FacilityState.lastUpdated` and the return doc of `getFacilityState()` both state that `lastUpdated == 0` means the facility has never been configured via `setAnnualGrowthRate`. In fact `issue` also checkpoints the facility and stamps `lastUpdated`, so zero only means the facility has never been checkpointed by anything. A facility can be issued against with the growth rate never set, leaving `lastUpdated` non-zero regardless.

3. `NFAT_INTEGRATION.md` states a strict one-to-one relationship between a PAU system and an NFAT facility, where a PAU stack serves exactly one facility and neither is shared. The facets impose no such constraint: both `NFATHaloFacet` and `NFATPrimeFacet` key all storage and rate limits by `facility` and support arbitrarily many facilities under a single PAU. Either the documentation overstates a deployment convention as an invariant, or the intended one-to-one constraint is unenforced in code.

Specification Partially Changed

In **Version 2**, mismatches 1 and 3 were corrected. Mismatch 2 remains: the NatSpec of `FacilityState.lastUpdated` still states that `lastUpdated == 0` means the facility was never configured via `setMaxAnnualGrowthRate`, while `issue` also checkpoints the facility and sets `lastUpdated`.

Acknowledged

Sky has created release `v1.14` and thus implicitly acknowledged the remaining mismatch.

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.