PALADIN
BLOCKCHAIN SECURITY

Final Report

Smart Contract Security Assessment

For Saphyre RFQ Settlement

DATE

29 April 2026

paladinsec.co

x.com/0xPaladinSec

Table of Contents

1 Disclaimer 3

2 Overview 4

2.1 Summary 4

2.2 Contracts Assessed 5

2.3 Findings Summary 6

3 Findings 8

3.1 RFQSettlement 8

1. Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

2. Overview

This report has been prepared for Saphyre RFQ Settlement's contracts on the Sei network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

2.1 Summary

Project Name	Saphyre RFQ Settlement
Project URL	https://saphyre.xyz/
Platform	Sei
Language	Solidity
Initial Commit	https://github.com/NIMA-Labs/rfq-contracts/tree/6c15ae631caa187ff4cca344a978da28a196d7e6
Resolution #1	https://github.com/NIMA-Labs/rfq-contracts/tree/26ff3d94135de05875773ce5afe71efbc337d81e

2.2 Contracts Assessed

CONTRACT	CONTRACT ADDRESS	LIVE CODE MATCH
RFQSettlement	0x6aE46fc725605f2eBb0126646f8318A069d328A2	Matched

2.3 Findings Summary

SEVERITY	FOUND	RESOLVED	PARTIALLY RESOLVED	ACKNOWLEDGED
GOVERNANCE	0	-	-	-
HIGH	0	-	-	-
MEDIUM	0	-	-	-
LOW	0	-	-	-
INFORMATIONAL	6	2	-	4
Total	**6**	**2**	**-**	**4**

ISSUE CLASSIFICATION

SEVERITY	DESCRIPTION
GOVERNANCE	Issues under this category are where the governance or owners of the protocol have certain privileges that users need to be aware of, some of which can result in the loss of user funds if the governance's private keys are lost or if they turn malicious, for example.
HIGH	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
MEDIUM	Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
LOW	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
INFORMATIONAL	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

RFQSettlement

ID	SEVERITY	SUMMARY	STATUS
01	INFO	Fee parameter is relayer-supplied and not signer-committed	ACKNOWLEDGED
02	INFO	`rescueTokens` is permissionless and allows public donation sweeping	ACKNOWLEDGED
03	INFO	Fee rounding can produce zero effective fee on small notionals	ACKNOWLEDGED
04	INFO	Fee cap constant is not centralized with protocol constants	RESOLVED
05	INFO	Deflationary or debasing tokens are not supported by settlement accounting	ACKNOWLEDGED
06	INFO	Setfeetreasury does not emit an event	RESOLVED

3. Findings

3.1 RFQSettlement

`RFQSettlement` is an atomic swap settlement contract that matches off-chain negotiated trades between a maker and taker by verifying their signed swap intents through `Permit2`, deducts a configurable protocol fee from the taker's input tokens, validates exact amount matching between parties, and distributes tokens to each participant with optional WETH-to-native unwrapping capability, ensuring all-or-nothing execution with reentrancy protection.

Privileged Functions

None

Issues & Recommendations

Issue #01 INFORMATIONAL

Fee parameter is relayer-supplied and not signer-committed

DESCRIPTION

The settlement fee (`feePips`) is provided at execution time via `RFQParams` and is not explicitly included in the signed maker/taker intent structs.

In the current design, the maker is still indirectly protected because `makerSwapIntent.outputAmount` must match `takerInput - fee` , but the fee rate itself is not independently committed by both parties in signed payloads.

RECOMMENDATION

If stronger fee consent guarantees are desired, include `feePips` (or `maxFeePips`) in the signed taker intent and validate it in `settleRFQ` .

RESOLUTION — ROUND 1 ACKNOWLEDGED

This is a design choice, and is documented in NatSpec.

Issue #02 INFORMATIONAL

`rescueTokens` is permissionless and allows public donation sweeping

DESCRIPTION

`rescueTokens(address token, address to)` is externally callable without access control. Any caller can move any accidental token/ETH balance held by the contract to an arbitrary destination.

RECOMMENDATION

Consider restricting `rescueTokens` to `onlyOwner`, or enforcing a fixed receiver (e.g., treasury/owner) to align with conservative operational controls.

RESOLUTION — ROUND 1 ACKNOWLEDGED

This is a design choice, and is documented in NatSpec.

Issue #03 INFORMATIONAL

Fee rounding can produce zero effective fee on small notionals

DESCRIPTION

Fee calculation uses integer division:

```
feeAmount = (takerSwapIntent.inputAmount * rfqParams.feePips) / PIPS
```

For sufficiently small input amounts, this can round down to zero even when `feePips > 0`. This behavior is consistent with integer math and the code already tolerates zero fee outflow, but it may differ from operational expectations if a non-zero fee is assumed for all settled orders.

RECOMMENDATION

Document this as accepted precision behavior, or enforce a minimum notional / minimum fee policy when `feePips` is non-zero.

RESOLUTION — ROUND 1 ACKNOWLEDGED

This is a design choice, and is documented in NatSpec.

Issue #04 INFORMATIONAL

Fee cap constant is not centralized with protocol constants

DESCRIPTION

`MAX_FEE_PIPS` is declared inside `RFQSettlement.sol`, while other shared protocol constants are maintained in `Constants.sol`.

RECOMMENDATION

Consider moving `MAX_FEE_PIPS` to `Constants.sol` for better centralization of protocol parameters and easier maintenance.

RESOLUTION — ROUND 1 RESOLVED

Issue #05 INFORMATIONAL

Deflationary or debasing tokens are not supported by settlement accounting

DESCRIPTION

Settlement assumes the contract receives exactly the requested token amounts from `Permit2` (`requestedAmount`) and then forwards exact outflows to maker, taker, and treasury.

For deflationary or rebasing tokens, the contract may receive less than requested, which will break downstream outflow expectations and cause settlement reverts.

RECOMMENDATION

Document that such tokens are unsupported, or add balance-before/balance-after accounting and adjust flow logic to operate on net received amounts.

RESOLUTION — ROUND 1 ACKNOWLEDGED

This is a design choice, and is documented in NatSpec.

Issue #06 INFORMATIONAL

Setfeetreasury does not emit an event

DESCRIPTION	`setFeeTreasury` is a state changing function that does not emit an event.
RECOMMENDATION	Consider emitting an event in the end of the function with the old and new address.

RESOLUTION — ROUND 1 RESOLVED


PALADIN
BLOCKCHAIN SECURITY