Zellic

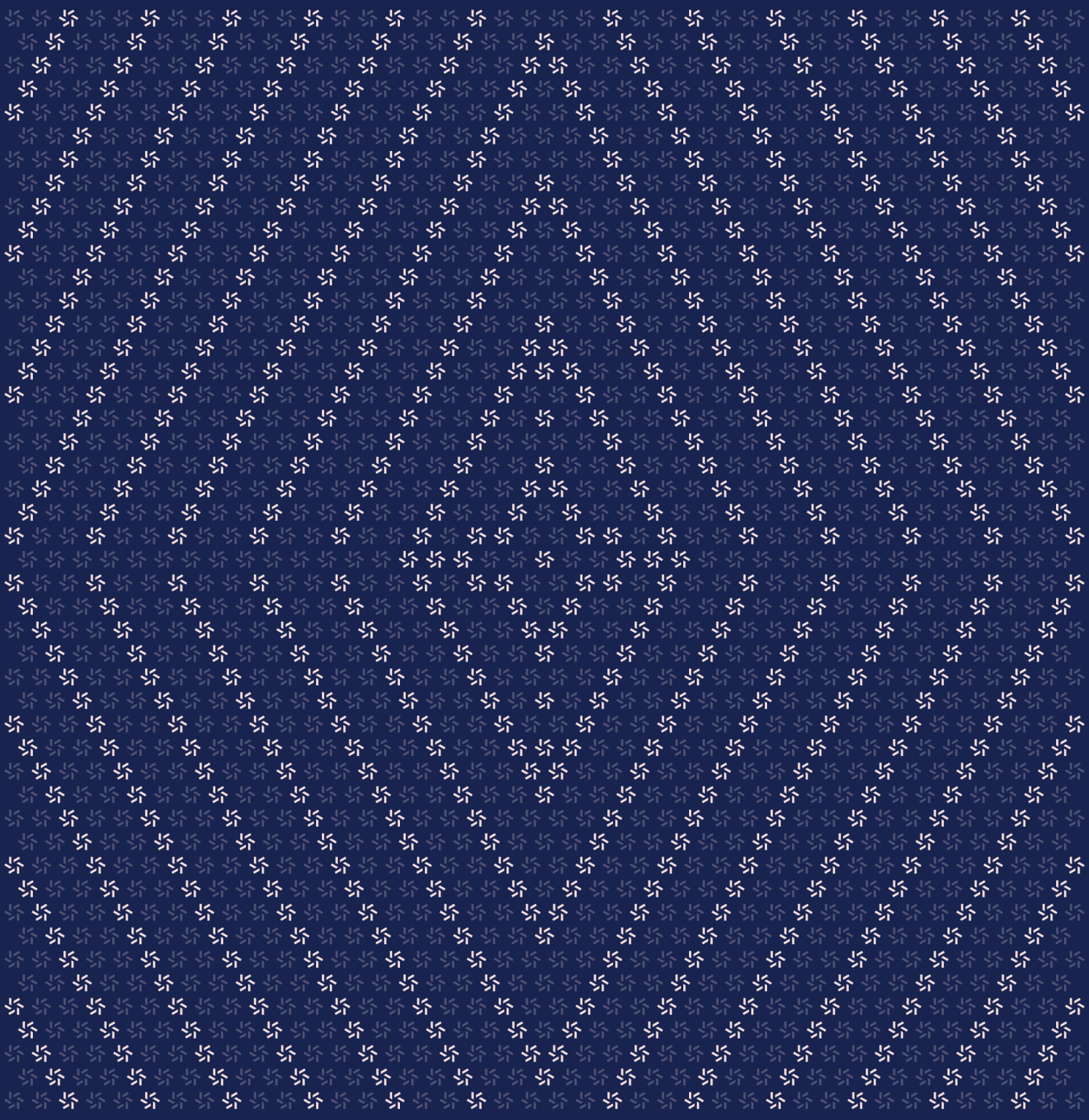
Prepared for
Nikola Madjarevic
Konstantin Jaredic
Miljan Rakita
Nima Labs

Prepared by
Quentin Lemauf
Chongyu Lv
Zellic

March 3, 2026

RFQ Protocol

Smart Contract Security Assessment



Contents

About Zellic **4**

1. Overview **4**

1.1. Executive Summary 5

1.2. Goals of the Assessment 5

1.3. Non-goals and Limitations 5

1.4. Results 5

2. Introduction **6**

2.1. About RFQ Protocol 7

2.2. Methodology 7

2.3. Scope 9

2.4. Project Overview 9

2.5. Project Timeline 10

3. Detailed Findings **10**

3.1. Unsigned fee rate allows relayer-side fee substitution 11

3.2. Permissionless rescue path is front-runnable and MEV-raceable 12

3.3. ETH transfers can fail and stop settlement and rescue 13

3.4. Test suite misses some settlement edge cases 14

4. System Design **14**

4.1. Component: RFQSettlement (Core Settlement Engine) 15

4.2. Component: Intent pairing and validation logic 16

4.3. Component: Permit2 authorization and token pull flow 17

5. Assessment Results 18

5.1. Disclaimer 19

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.



1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Nima Labs on March 2nd, 2026. During this engagement, Zellic reviewed RFQ Protocol's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Is the settlement contract for off-chain coordinated swaps secure against unauthorized asset movement and loss of user funds?
- Are Permit2 witness payloads (maker and taker intents) constructed and validated correctly end to end?
- Can Permit2 signature malformation, malformed witness/type strings, or replay-style edge cases bypass settlement authorization checks?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

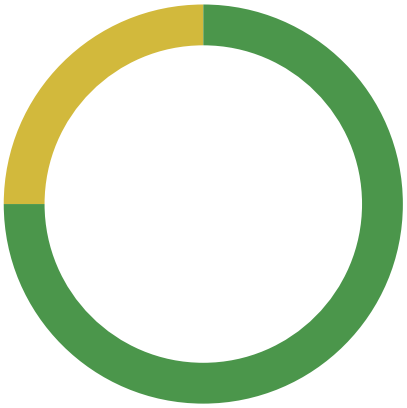
Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped RFQ Protocol contracts, we discovered four findings. No critical issues were found. One finding was of medium impact and three were of low impact.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	1
Low	3
Informational	0



2. Introduction

2.1. About RFQ Protocol

Nima Labs contributed the following description of RFQ Protocol:

> RFQ Protocol.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

2.3. Scope

The engagement involved a review of the following targets:

RFQ Protocol Contracts

Type	Solidity
Platform	EVM-compatible
Target	rfq-contracts
Repository	https://github.com/NIMA-Labs/rfq-contracts ↗
Version	`6c15ae631caa187ff4cca344a978da28a196d7e6`

Programs	`Common.sol`
	`Constants.sol`
	`RFQSettlement.sol`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 1.5 person-days. The assessment was conducted by two consultants over the course of one calendar day.

Contact Information

The following project manager was associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io

The following consultants were engaged to conduct the assessment:

Quentin Lemauf
Engineer
quentin@zellic.io

Chongyu Lv
Engineer
chongyu@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

March 2, 2026	Start of primary review period
March 2, 2026	End of primary review period

3. Detailed Findings

3.1. Unsigned fee rate allows relayer-side fee substitution

Target	RFQSettlement		
Category	Integration Risks	**Severity**	Medium
Likelihood	Medium	**Impact**	Medium

Description

The `settleRFQ` fee calculation depends on `rfqParams.feePips`, but `feePips` is not included in either side's signed EIP-712 witness payload. The contract computes `feeAmount` as `takerSwapIntent.inputAmount * rfqParams.feePips / PIPS` and then derives `quoteAmount` from that result. While maker and taker intents are signed, `feePips` only exists in `RFQParams`, so it remains caller-controlled at execution time.

Because Solidity integer division truncates toward zero, different fee rates can collapse to the same `feeAmount` for a given notional. When `takerInputAmount * |feePips_A - feePips_B| < PIPS`, both fee rates produce identical rounded output, so amount checks still pass. A permissionless relayer can therefore submit a `feePips` value different from the negotiated value without invalidating settlement.

For example, a maker and taker negotiate `feePips = 100` and `takerInputAmount = 500`. Since `500 * 100 / 1,000,000 = 0`, the signed amounts are consistent with a zero rounded fee. A relayer can submit `feePips = 1`, which also yields `500 * 1 / 1,000,000 = 0` — settlement succeeds, the protocol fee remains zero, and `RFQSettled` records `feePips = 1` instead of the negotiated `100`.

Impact

Relayer-controlled fee substitution can reduce fees paid to `feeTreasury` while preserving on-chain settlement validity. Additionally, `RFQSettled` can publish a `feePips` value that does not match the negotiated off-chain fee rate.

Recommendations

Include `feePips` in the signed witness payload so both counterparties explicitly authorize the exact fee rate used on chain.

Remediation

This issue has been acknowledged by Nima Labs.

3.2. Permissionless rescue path is front-runnable and MEV-raceable

Target	RFQSettlement		
Category	Protocol Risks	**Severity**	Low
Likelihood	High	**Impact**	Low

Description

The `rescueTokens` function is permissionless and allows any caller to transfer the full contract balance of a selected asset to an arbitrary recipient.

```
/// @notice withdraw accidental donations
function rescueTokens(address token, address to) external nonReentrant {
    if (address(0) == token) {
        payable(to).transfer(address(this).balance);
    } else {
        IERC20(token).safeTransfer(to,
    IERC20(token).balanceOf(address(this)));
    }
}
```

If a legitimate user broadcasts a rescue transaction, a searcher can copy the call, replace `to`, and front-run. Since the function transfers the entire available balance, the front-running transaction captures all recoverable funds and the original transaction executes with zero value.

Impact

Accidental donations are captured by third parties before intended recovery transactions are executed.

Recommendations

Restrict rescue authority to a trusted role.

Remediation

This issue has been acknowledged by Nima Labs.

3.3. ETH transfers can fail and stop settlement and rescue

Target	RFQSettlement		
Category	Integration Risks	**Severity**	Low
Likelihood	Medium	**Impact**	Low

Description

The contract uses `payable(recipient).transfer(amount)` for native ETH outflows in both the WETH unwrap settlement path and the ETH rescue path. A fixed 2,300 gas stipend is forwarded by `transfer`, which is not sufficient for many modern contract-wallet receive handlers. If the recipient contract requires more gas in `receive` or `fallback`, the transfer reverts and the parent operation fails.

In the settlement path, this can cause valid RFQs with `unwrap=true` to fail purely due to recipient implementation details. The same limitation can also block ETH rescue operations directed to contract recipients.

Impact

Valid settlements that require WETH-to-ETH delivery can fail for contract recipients. Rescue operations can fail for contract recipients with nontrivial receive logic.

Recommendations

Replace `transfer` with `call{value: amount}("")`, and require a successful return value.

This is also the recommended forward-compatible pattern, because `transfer` and `send` are deprecated as of Solidity 0.8.31 (see the Solidity documentation ↗).

Remediation

This issue has been acknowledged by Nima Labs, and a fix was implemented in commit `db25973e` ↗.

3.4. Test suite misses some settlement edge cases

Target	Project-wide		
Category	Protocol Risks	**Severity**	Low
Likelihood	N/A	**Impact**	Low

Description

The project includes a solid baseline test suite (10 passing tests) that covers valid settlement flows and several revert paths. However, it still misses some edge-case scenarios identified during this audit.

In particular, as of the time of writing, the tests do not explicitly exercise

- relayer-side `feePips` substitution in truncation-equivalent ranges,
- ETH payout failure when recipient contracts require more than the 2,300 gas stipend used by `transfer` and
- access-control behavior for `setFeeTreasury` from unauthorized callers and successful owner updates.

Impact

This has several impacts.

- Edge-case behavior may regress undetected during refactors.
- Economic behavior (fee collection and reporting) can diverge from protocol expectations without an immediate test signal.
- Integration/liveness assumptions for contract recipients can fail in production despite passing test runs.

Recommendations

Expand the suite with targeted tests for fee-parameter edge cases, fee-rounding boundary values, contract-recipient ETH payout failures, and privileged function authorization paths.

Remediation

This issue has been acknowledged by Nima Labs.

4. System Design

This provides a description of the high-level components of the system and how they interact, including details like a function's externally controllable inputs and how an attacker could leverage each input to cause harm or which invariants or constraints of the system are critical and must always be upheld.

Not all components in the audit scope may have been modeled. The absence of a component in this section does not necessarily suggest that it is safe.

4.1. Component: RFQSettlement (Core Settlement Engine)

Description

RFQSettlement is the only stateful contract in scope. It executes bilateral RFQs that are negotiated off chain and settled on chain in one atomic transaction.

The contract combines three roles: settlement execution, fee routing, and operational controls.

Settlement is centered on `settleRFQ(RFQParams)`, which validates a maker/taker pair, computes fee and net quote amounts, pulls both parties' inputs through Permit2, and then distributes the three outflow legs.

- Settlement flow — validate pair → compute `feeAmount` / `quoteAmount` → pull maker/taker inputs → pay treasury, maker, and taker.
- Fee handling — quote-side fees are routed to `feeTreasury` and bounded by `MAX_FEE_PIPS`.
- Token delivery mode — WETH outputs can optionally be unwrapped and sent as native ETH.
- Admin/update path — the owner can rotate `feeTreasury` via `setFeeTreasury`.
- Recovery path — `rescueTokens` can move full ETH/ERC-20 balances to a caller-selected recipient.

Invariants

Core safety properties are straightforward and enforced directly in `settleRFQ` and constructor checks.

- Address validity — `permit2`, `weth9`, `owner`, `feeTreasury`, and runtime token addresses must be nonzero.
- Fee bounds — `feePips` must satisfy `0 < feePips <= MAX_FEE_PIPS`.
- Fee math consistency — `feeAmount = taker.inputAmount * feePips / PIPS` and `quoteAmount = taker.inputAmount - feeAmount`.
- Direction/amount consistency — maker and taker intents must be opposite and amount-compatible before outflows execute.

Attack surface

Most risk comes from open execution and external dependencies rather than complex internal state.

- Permissionless caller model — anyone can call `settleRFQ`, so the caller controls submission timing and RFQ packaging.
- External dependency trust — correctness depends on Permit2 validation plus ERC-20/WETH behavior.
- Unsigned runtime fee input — `feePips` is calldata-provided at execution and not part of current (as of the time of writing) signed witness payloads.
- ETH outflow liveness — native ETH delivery uses `transfer`, which may fail for some contract recipients.
- Rescue raceability — `rescueTokens` is permissionless and has a full balance , so recovery attempts can be front-run in public mempools.

4.2. Component: Intent pairing and validation logic

Description

One `RFQParams` payload containing `makerSignedSwapIntent`, `takerSignedSwapIntent`, and `feePips` is received by `settleRFQ`. From those inputs, the contract derives effective trade values and decides whether the pair is executable.

This layer acts as the on-chain matcher of the system.

- Input structure — one maker intent, one taker intent, and one fee parameter.
- Derived values — `feeAmount`, `quoteAmount`, and `baseAmount` are computed from taker values plus `feePips`.
- Pairing logic — the maker and taker must be opposite in token direction and consistent in amount equations.
- Counterparty mode — the maker can either bind to one taker or stay open with `counterparty = address(0)`.

Invariants

Pair acceptance is constrained by a set of hard rules that must all hold simultaneously.

- Counterparty rule — if the maker `counterparty` is nonzero, it must equal the taker signer.
- Token-direction rules — `maker.inputToken == taker.outputToken` and `maker.outputToken == taker.inputToken`, with identical token pairs rejected.
- Amount rules — `maker.inputAmount == taker.outputAmount` and `maker.outputAmount == taker.inputAmount - feeAmount`.
- Nonzero trade outputs — `quoteAmount` and `baseAmount` must both be greater than zero.
- Fee range rule — `feePips` must stay within configured bounds.

Attack surface

Risk in this component mainly comes from open submission, arithmetic boundaries, and trust in off-chain packaging.

- Caller packaging freedom — any relayer can choose which maker/taker pair and `feePips` value to submit.
- Open maker orders — `counterparty = address(0)` intentionally allows any compatible taker to fill.

4.3. Component: Permit2 authorization and token pull flow

Description

Settlement authorization relies on Permit2 witness signatures from both participants. The contract reconstructs each side's permit payload from signed fields, rebuilds the witness hash from the signed intent, fixes transfer destination to `address(this)`, and then calls `permit2.permitWitnessTransferFrom(...)`.

This creates a custody-first execution model where authorization and token pulls happen before value distribution.

- Input preparation — build `PermitTransferFrom` from signed `token`, `amount`, `nonce`, and `deadline`.
- Witness binding — rebuild the witness hash from `MakerSwapIntent` and `TakerSwapIntent`.
- Transfer target — the request pulled tokens directly into the settlement contract.
- Authorization call — verify signature and execute pull through Permit2.
- Execution order — validate pair → pull both inputs → route fee and output transfers.

Invariants

Core correctness here depends on signature integrity, amount consistency, and ordering guarantees.

- Signature binding — maker and taker signatures must commit to their exact witness payloads.
- Requested amount consistency — each requested pull amount must match the signed input amount.
- Destination consistency — pulled assets are always directed to the settlement contract.
- Replay controls — nonce and deadline checks are enforced by Permit2 for both signers.
- Ordering guarantee — authorization and custody must succeed before any fee or user outflow.

Attack surface

Risk is concentrated in external trust boundaries and serialization correctness rather than in local state complexity.

- Permit2 dependency — signature validation and replay protection are externalized to Permit2.

5. Assessment Results

During our assessment on the scoped RFQ Protocol contracts, we discovered four findings. No critical issues were found. One finding was of medium impact and three were of low impact.

The issues found in this audit are mostly edge cases, with limited direct fund-loss impact in the design. All of our findings should be fixed, but the big item to address is fee-parameter commitment during settlement, plus hardening of the rescue and ETH transfer paths.

Overall, we consider the codebase close to deployment readiness once the reported fixes are applied. We recommend adding focused regression tests for the identified edge cases and performing an internal verification pass before release.

5.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.