SkirkLabs

ZEROLIQUIDITY

Token Launchpad & Orbit contract Review

FOUNDRY BASED SECURITY AUDIT

AUGUST, 2026

1 Executive summary

ZeroLiquidity is a token launchpad built on Uniswap v4. The safety claim is simple and structural: when a token launches, 100% of its supply is deposited single sided into a pool owned by the launcher contract, and there is no function anywhere in the protocol that removes that liquidity. Every launch includes a small genesis buy that is burned on the spot, which puts a permanent floor under the price.

Over the whole auditing process, **we never found a Critical, High, or Medium severity vulnerability**, in development, in testing, or in the deployed contracts. Mainly because the codebase follows good practices of simple, modular code(less is more). Leaving a small scope: one external entry point for launching, a fee claiming function, and the metadata setters on the token. What we did find, and what this report lists in full, were **Low severity issues and informational notes**: flows that were not optimal, design decisions we reversed after arguing about them, integration traps, and tooling problems. Seven of the eight Low findings are fixed in version 3; the eighth is mitigated, with the residual risk documented.

	Critical	High	Medium	Low	Informational
Found	0	0	0	8	5

All eight Low findings are resolved or mitigated. Of the five informational notes, three are resolved and two are accepted constraints, one of which we actively monitor.

2 Scope and methodology

This review covers the on-chain protocol: the launcher (`ZeroLiquidity` and its internal layers), the launched token (`ZeroToken`), the tandem launcher for official satellite pools (`ZeroOrbit`), and the read only aggregation sidecar (`ZeroLens`). Findings came from three places:

- **Continuous testing.** 73 tests across seven suites, including invariant tests in both pool orientations, fuzzing, and a fork suite that replays the payment flow against the real wrapped native contracts and pool managers of four production networks.
- **Self review.** Before each redesign we tried to break it on paper. Several findings below were caught this way and never made it into shipped code.
- **Live operation.** Test launches and release ceremonies on public networks. This is where the tooling and infrastructure findings came from.

3 Attack surface

The launcher has one state changing entry point for users, `launch()`, plus `claimFees()` for collecting accrued trading fees. Administration works through a relay: a cold key signs typed EIP-712 actions offline, any wallet can submit them, and each signature is one shot with a deadline. The token adds creator gated metadata setters on top of a standard, unmodified ERC-20. There is no upgradeability, no hooks, no privileged launch path, and no function that removes liquidity. Fees pay out directly from the pool manager, so the launcher never holds user funds between transactions.

Severity	Definition used in this report
Critical / High	Loss or freezing of user funds, theft of liquidity, a broken core invariant. None found at any point.
Medium	Griefing or degraded guarantees requiring unusual preconditions. None found.
Low	Unoptimized or avoidably complex flows, design weaknesses we caught and corrected before they could affect users, failure modes limited to a reverted transaction.
Informational	Observations, constraints, and lessons with no direct security impact.

4 Findings, Low severity (resolved or mitigated)

L-1 Wrapped ether launches required user side wrapping and approvals LOW

Issue. In early versions, launching against wrapped ether meant the creator first had to wrap ETH themselves and then grant the launcher an ERC-20 allowance. Two extra transactions, an open allowance sitting around, and extra gas on the most common path in the product. In hindsight the wrapping step should always have been the contract's job.
Resolution. FIXED (v2d) We redesigned the payment flow. A launch is now a single plain transaction carrying raw ether in `msg.value`. Inside the settlement callback the launcher wraps exactly the amount the genesis buy consumed and refunds the rest at the very end of the transaction. Before shipping this we wrote a fork suite that checks the real wrapped native contracts on all four target networks behave one to one under this flow. Version 3 went one step further and added raw ether pools where no wrapped token is involved at all.

L-2 Settlement ordering in the genesis callback LOW

Issue. The genesis floor buy withdraws real tokens from the pool manager and sends them to the burn address, which only works if the pooled supply has already been physically settled. Transient accounting does not cover ERC-20 balances. An early version ran the buy first, and the launch reverted against itself. Worst case here was always a clean revert of the whole launch, and it never reached a deployment.
Resolution. FIXED (v1, before deploy) The callback settles the pooled supply before the buy runs. The ordering is written down as a rule in the source and mirrored in the test harness so nobody reintroduces it.

L-3 Stock settlement helper incompatible with non standard ERC-20s LOW

Issue. Uniswap's settlement helper expects a boolean return from `transferFrom`. Some widely used stablecoins return nothing, so any settlement on their side would revert.
Resolution. FIXED (v2) We settle quote tokens by hand (`sync`, safe transfer, `settle`) in a way that tolerates both return conventions. We also made it policy that quote currencies are individually allowlisted by the cold key, and only standards compliant assets get blessed.

L-4 Fee burn mode was originally chosen per claim LOW

Issue. One of our favorite features lets a creator renounce the token side of their own trading fees forever. The first design made that a flag passed at each claim. Since the operator key may claim on a creator's behalf as a convenience, a compromised operator key could in theory have forced the burn flag on someone else's claim. It needed a stolen

privileged key to matter, and no production launch was ever exposed, but we did not like the smell of it.
Resolution. FIXED (v2c) The choice became a one time commitment made at creation and stored immutably in the launch record. The claim path carries no mode parameter at all, so nothing a caller passes can change the routing. This also made the feature better: the commitment is now readable on chain, which is what lets the site show it as a trust badge.

L-5 A privileged fee free launch path existed in early versions — LOW

Issue. An admin only launch function let us launch without paying the creation fee. It was convenient for testing and a bad idea for trust. A second door into the system has to be audited forever.
Resolution. FIXED (v2) We deleted it. `launch()` is the only way in, and our own operator key pays the fee and the floor budget like any stranger. A dedicated test keeps it that way.

L-6 Event log discovery does not scale on fast chains — LOW

Issue. We planned per creator discovery as an event log filter. On a network with sub second blocks and a capped log scan window, the cost of a sweep grows with the age of the chain. Measurements showed the approach simply failing on one of our target networks.
Resolution. FIXED (Lens v2) The read sidecar got a paginated in-EVM scan whose cost is capped per call by construction: the limit counts positions scanned rather than matches, so no address can force an unbounded walk. The edge cases are pinned by tests, including that a zero limit must not look like completion and that offset arithmetic clamps instead of wrapping.

L-7 Creator identity is claimed, not proven — LOW

Issue. Version 3 separates the launch owner from the payer, which enables launch on behalf and third party integrations. The flip side is that anyone can launch a token and name any address as its creator, which is an impersonation surface.
Resolution. MITIGATED (v3) The launch event emits the actual payer next to the claimed creator. The interface flags mismatches and never treats the creator field alone as proof of identity. We accept the residual risk and document it. Comparable launchers made the same call.

L-8 The launch time floor budget invited over commitment — LOW

Issue. Creators could size the genesis floor buy themselves. We eventually concluded this was a trap dressed up as a feature: the floor purchase is burned, so a larger buy just parks the creator's money in a position nobody can ever touch again.
Resolution. FIXED (v3) The genesis buy is now always the pair's configured minimum, a small symbolic burn that seeds the price floor. The old parameter stays in the ABI for compatibility and is ignored. We lowered the minimums at the same time.

5 Findings, Informational

I-1 Contract size runs close to the EIP-170 ceiling — INFORMATIONAL

Unoptimized, the launcher does not fit in the 24,576 byte deployment limit. Optimized, it ships with under 1,000 bytes of headroom. We treat this as an architectural constraint: new capability goes into stateless sidecars (the read lens, the tandem launcher) that can be deployed or replaced at any time without touching the core, and events stay slim. **ACCEPTED**

I-2 One network's wrapped native token is upgradeable — INFORMATIONAL

On one supported network the canonical wrapped native token is a governance upgradeable proxy, so its deposit semantics could in principle change under us. Our review concluded the worst case is a launch that reverts, a denial of one transaction, with no path to fund loss. Version 3's raw ether pools avoid the wrapper entirely for native launches, and the fork suite acts as a standing tripwire. **ACCEPTED, MONITORED**

I-3 Fee growth counters wrap by design — INFORMATIONAL

Uniswap's cumulative fee growth counters overflow intentionally, so deltas must be computed with wrapping subtraction. The read sidecar handles this explicitly, and its predictions are tested to equal actual claim payouts to the wei, in both pool orientations and under the burn commitment. **RESOLVED IN DESIGN**

I-4 A semantics only change silently broke an integrator — INFORMATIONAL

The payment redesign in L-1 changed how `launch()` expects to be paid without changing its ABI. An integration using the old flow type checked fine and reverted at runtime. The lesson stuck: a semantics only change now gets an explicit integration handoff, precisely because no tooling will catch it. **PROCESS RULE ADOPTED**

I-5 Deployment and verification tooling hurdles — INFORMATIONAL

Three operational issues turned up in the release tooling, never in the protocol. The leftover gas recovery step in our deploy scripts originally trusted a simulation time balance and under reserved real gas. The source verification tooling briefly drifted from the deploy scripts' constructor arguments after a parameter retune, which we caught in a preflight review before it could fail any verification. And explorer API rate limits could starve the second of two back to back verifications. All three are fixed. The current release flow reads its inputs directly from broadcast records and verifies the whole surface with one command. **FIXED**

6 Hurdles worth recording

6.1 Designing under the size ceiling

The 24KB deployment limit forced the layered architecture (state, machinery, and the external surface as separate inheritance layers) and pushed us toward sidecars. In the end the constraint helped: everything outside the trust surface, meaning reads, aggregation, and tandem launches, can be replaced without redeploying anything that holds funds.

6.2 Uniswap v4 conventions

v4 inverts v3's sign convention for exact input versus exact output swaps, settles through transient accounting that does not cover real ERC-20 balances, and stops at a price limit instead of reverting. Each of these is a documented warning in our source today because each one cost us a debugging session once.

6.3 The curve had to be measured, not assumed

The anti whale band curve was validated with a dedicated measurement harness: allocation shapes, self deepening behavior, dump asymmetry, and entry quality across geometries. The default we recommend, 12 bands over roughly a 100,000x range, comes out of those measurements.

6.4 Real infrastructure is part of the threat model

Sub second block times break log scans. Explorers lag, rate limit, and sometimes report failure after a submission already succeeded. RPC providers disagree about receipt polling. The version 3 tooling assumes all of this from the start: bounded on chain reads instead of log sweeps, one command verification fed from broadcast records, and a habit of checking chain state before retrying any broadcast that looks failed.

7 Assurance summary

Suite	Tests	What it pins
Core launcher	18	Launch invariants in both orientations, payment rules (overpayment and underpayment both revert), floor rule, whale curve economics, round trip floor hold, fee split and gating, cold key governance and rotations, single entry rule, preview accuracy
Read sidecar	10	Fee predictions equal actual claims to the wei, pagination boundedness and edge cases
Tandem launcher	6	Money conservation across both pools, parity rule, single claim surface, commitment propagation, the contract rests at zero balances
Interface proof	1	The full flow driven purely through the public interface files, so an ABI mismatch fails in CI instead of in production
Byte parity	1	Annotated sources and published sources compile to identical bytecode
Fork parity	24	Six tests on each of four production networks: live bytecode versus current source, real wrapper semantics, raw ether launches against real pool managers
Curve study	13	The measurement harness behind the recommended band geometry

Beyond the suites, the tandem claim path was proven against a live public testnet deployment before shipping anywhere real, and the first production launch's token was source verified so explorers automatically match every later token by identical bytecode.

8 Disclaimer

This report is a good faith account of our own development process. It is not an independent security audit and it is not investment advice. The protocol's guarantees are structural claims about its own code: liquidity locked by construction, a burned price floor, immutable fee commitments. Every one of them can be checked against the verified on chain sources. Do your own research.

SkirkLabs ▪ August 2026
Author: Kevin Vuillaume