

Technical audit report

Line-by-line security review of the in-scope code. Each finding carries the technical detail for your engineers and a plain-terms explanation for everyone else.

The subject of this sample, the Lumenswap Protocol, is a fictional decentralized exchange invented for illustration. Names, figures and findings are examples of format and depth, not real results.

1. Engagement summary

Client	Lumenswap Labs (fictional)
Codebase	lumenswap-core at commit 4be1f02 (example)
Scope	Router, Vault, OracleAdapter, FeeSplitter, RewardStaker, LumenToken (4,120 lines of Solidity 0.8.x, example figure)
Out of scope	Front-end, third-party bridge, off-chain keeper bots
Review window	Ten working days, two senior auditors
Methods	Manual line-by-line review, AI-assisted cross-checks, fuzzing, invariant testing, fork-based exploit reproduction

2. Executive summary

We found twelve issues: one critical, two high, four medium, one low and four informational. Every actionable finding was fixed by the team and re-verified by us against the original exploit, except one medium accepted with a documented plan. The critical finding allowed complete draining of pooled funds and was reproduced on a fork before disclosure. After remediation, the codebase holds against every attack we constructed.

3. Severity definitions

Severity	Meaning
Critical	Direct loss or freezing of user funds, exploitable by anyone
High	Loss under realistic conditions, or trust assumptions broken
Medium	Bounded loss, or requires unusual conditions to exploit
Low	Best-practice gap with indirect or operational risk
Informational	Code quality, gas and hardening notes; no direct risk

4. Findings overview

ID	Title	Severity	Status
TA-01	Reentrant withdrawal drains the vault	Critical	Fixed, verified
TA-02	Stale oracle price accepted as fresh	High	Fixed, verified
TA-03	Swaps lack slippage and deadline bounds	High	Fixed, verified
TA-04	Unchecked return value on legacy tokens	Medium	Fixed, verified

ID	Title	Severity	Status
TA-05	Fee math loses precision for small trades	Medium	Fixed, verified
TA-06	Initialization can be front-run	Medium	Fixed, verified
TA-07	Unbounded reward loop can jam claims	Medium	Accepted, mitigation planned
TA-08	Privileged changes emit no events	Low	Fixed, verified
TA-09..12	Gas and code-quality notes	Informational	Adopted

TA-01. Reentrant withdrawal drains the vault

CRITICAL

Location

Vault.sol, withdraw(), lines 141-158 (example)

Description

withdraw() transfers tokens before writing the caller's updated balance. A token with a transfer hook can re-enter withdraw() while the stored balance is still the old value, repeating the transfer until the vault is empty.

In plain terms

The vault pays out before it updates its books. A malicious token can keep asking to be paid against the old balance, like cashing the same cheque repeatedly before the bank records it. Everything in the pool could be stolen in one transaction.

Impact

Total loss of pooled user funds by any attacker, no special access required.

Proof of concept

A 40-line attacker contract recursively withdrew the full vault balance in a single transaction on a mainnet fork.

Recommendation

Write state before external calls (checks-effects-interactions) and add a reentrancy guard to every balance-touching entry point.

Resolution

Fixed in the remediation commit. We re-ran the original exploit and the fuzzing suite against the patch: no path remains.

TA-02. Stale oracle price accepted as fresh

HIGH

Location

OracleAdapter.sol, latestPrice(), lines 52-60 (example)

Description

latestPrice() returns the feed's last answer without checking its timestamp against the feed heartbeat. During a feed outage the protocol keeps trading on the frozen price.

In plain terms

If the price feed stops updating, the exchange keeps using the last known price as if it were current. In a moving market, traders could swap at yesterday's price and drain value from the pool.

Impact

Mispriced swaps against liquidity providers whenever the upstream feed stalls; loss scales with market movement during the outage.

Proof of concept

Simulated a 30-minute feed halt on a fork while moving the market 12%: the pool accepted swaps at the stale rate throughout.

Recommendation

Reject answers older than the heartbeat plus a grace window; pause swaps on staleness rather than defaulting through.

Resolution

Fixed and verified: the halt simulation now pauses the pool within one block.

TA-03. Swaps lack slippage and deadline bounds

HIGH**Location**

Router.sol, swapExact(), lines 88-97 (example)

Description

swapExact() accepts a minimum-out of zero and no deadline. Pending transactions can be sandwiched: an attacker moves the price before the victim's swap executes and restores it after, pocketing the difference. Stuck transactions can also execute much later at a worse price.

In plain terms

An order without a price floor or expiry is a blank cheque: bots that watch pending transactions can worsen your price on purpose and keep the difference.

Impact

Every user swap is exposed to MEV sandwiching; measurable, ongoing value leak rather than a single dramatic theft.

Proof of concept

A sandwich bot script on a fork extracted 2.4% from a sample swap with zero minimum-out (example figure).

Recommendation

Require a non-zero minimum output and a deadline on every swap; have the front-end set sane defaults.

Resolution

Fixed and verified; the sample sandwich now reverts the victim transaction instead of filling at the worse price.

TA-04. Unchecked return value on legacy token transfers

MEDIUM**Location**

Router.sol, _payOut(), line 203 (example)

Description

transfer() return values are ignored. Legacy tokens that signal failure by returning false instead of reverting can fail silently while the router's accounting proceeds as if payment succeeded.

In plain terms

Some older tokens report failure quietly instead of stopping the transaction. The router does not listen for that signal, so it can mark a payment as done when no tokens actually moved.

Impact

Accounting drift and stuck balances for the affected token pairs.

Proof of concept

Deployed a false-returning mock token on a fork: the router recorded a successful payout while the recipient received nothing.

Recommendation

Use SafeERC20 wrappers for every token interaction.

Resolution

Fixed and verified across all token call sites.

TA-05. Fee math loses precision for small trades

MEDIUM**Location**

FeeSplitter.sol, split(), lines 61-70 (example)

Description

The 70/30 fee split divides before multiplying, so integer division truncates both legs on small amounts and the remainder silently stays in the splitter contract instead of reaching either beneficiary.

In plain terms

For tiny trades the math rounds both shares down and the leftover dust piles up in a contract nobody can withdraw from. Small individually, permanent and growing in aggregate.

Impact

Slow, unrecoverable value accumulation in the splitter; misreported revenue for both beneficiaries.

Proof of concept

Fuzzing found amounts where up to 2 wei per trade was stranded; projected across example volume that is a steady leak.

Recommendation

Multiply before dividing, assign the remainder explicitly to one beneficiary, and add an invariant test that the legs always sum to the input.

Resolution

Fixed and verified; the new invariant runs in CI.

TA-06. Initialization can be front-run

MEDIUM

Location

Vault.sol, initialize(), lines 20-31 (example)

Description

The upgradeable vault's initialize() is callable by anyone after deployment. A watcher can call it first, become owner, and set a hostile fee recipient before the team's own initialization lands.

In plain terms

Between deploying the contract and configuring it, there is a gap where a stranger can claim the owner seat first. Deployment order on a public chain is a race, and this code loses it.

Impact

Full takeover of a freshly deployed vault; recoverable only by redeploying, embarrassing at best on mainnet.

Proof of concept

On a local fork, a bot contract initialized the vault in the same block as deployment, ahead of the deployer's transaction.

Recommendation

Initialize atomically with deployment through the proxy constructor's calldata, and add an initializer guard.

Resolution

Fixed and verified; deployment scripts now deploy and initialize in one transaction.

TA-07. Unbounded reward loop can jam claims

MEDIUM

Location

RewardStaker.sol, claimAll(), lines 112-126 (example)

Description

claimAll() iterates every historical reward epoch for the caller. A long-lived account eventually exceeds the block gas limit and can never claim through this path again.

In plain terms

The claim function re-reads a user's whole history every time. Stay staked long enough and your own history grows too heavy to carry, locking the claim button for you.

Impact

Long-term stakers, the protocol's best users, can be locked out of rewards; funds are not lost but require a migration to recover.

Proof of concept

Simulated 400 epochs for one account: claimAll() exceeded the block gas limit (example figure).

Recommendation

Track a per-user accumulator instead of iterating epochs, or add a paginated claim path.

Resolution

Accepted for launch with a documented cap on epoch count and an accumulator refactor scheduled next release. We re-tested the cap.

TA-08. Privileged changes emit no events

LOW

Location

FeeSplitter.sol and OracleAdapter.sol, all setters (example)

Description

Owner-only setters for fee recipients and price feeds change critical parameters without emitting events, leaving monitoring and users blind to configuration changes.

In plain terms

When an admin changes something important, the contract should announce it publicly. These changes happen silently, so nobody watching the chain would notice a hostile or fat-fingered change.

Impact

Slower incident detection; weaker audit trail for governance.

Proof of concept

Not exploit-driven; verified by review and a monitoring dry run that missed a test parameter change.

Recommendation

Emit a dedicated event from every privileged setter; alert on them.

Resolution

Fixed and verified; the monitoring dry run now catches every change.

5. Informational notes (TA-09..12)

ID	Note
TA-09	Cache repeated storage reads in memory in Router loops; measurable gas saving on hot paths
TA-10	Replace revert strings with custom errors for cheaper, clearer failures
TA-11	Mark never-changing config immutable; set at construction
TA-12	Use calldata instead of memory for read-only array parameters

6. Verification statement

After the team's remediation commit, we re-executed every proof of concept, the full fuzzing corpus and the invariant suite. No fixed finding reproduces; the accepted finding's cap holds under test. The public attestation for this engagement references the audited and the remediated commit hashes.