7 Audit Passes, 35 Findings, 0 Compromises: How We Hardened VibeSwap's Smart Contracts

Seven is a lucky number. Seven days in a week. Seven notes in a scale. And now — seven full security audit passes across every smart contract in the VibeSwap protocol.

Here's what we found, what we fixed, and what we learned about building secure DeFi from scratch.

What Is VibeSwap?

VibeSwap is an omnichain DEX built on LayerZero V2 that eliminates MEV through commit-reveal batch auctions with uniform clearing prices. The protocol spans 25+ upgradeable smart contracts covering AMM mechanics, cross-chain routing, governance, incentive distribution, identity, compliance, and quantum-resistant security.

The codebase is built on Solidity 0.8.20 with OpenZeppelin v5.0.1, using UUPS proxy patterns and Foundry for testing.

Before the audit marathon began, we had 476 passing tests. By the end: **550 tests, 0 failures, 35+ vulnerabilities fixed across 14 commits.**

The Seven Passes

Pass 1: Deploy Scripts & Configuration

The boring stuff that would have ruined launch day.

We started where most auditors don't — the deployment scripts. Good thing we did:

- `CommitRevealAuction.initialize()` requires 3 parameters. Both deploy scripts passed 2. **Every deployment would have reverted on-chain.** Dead on arrival.
- `setGuardian()` was called on `VibeAMM`, which doesn't have that function. It belongs on `VibeSwapCore`.
- Emergency pause logic called `setGlobalPause()` on the AMM — a function that doesn't exist. The emergency brake was disconnected.

We also added post-deployment verification: EOA requirements, rate limits, guardian assignment, flash loan protection, TWAP validation, and router authorization — all checked automatically after deploy.

Lesson: Your contracts can be perfect and still fail at the last mile. Audit the scripts.

Pass 2: Core Contract Deep Dive

Where tokens actually move.

This pass focused on the contracts that handle real value — IncentiveController, VolatilityInsurancePool, StablecoinFlowRegistry, and FederatedConsensus.

The worst finding: **IncentiveController was silently swallowing failed deposits.** When routing fees to the volatility insurance pool via a low-level call, a failure returned `false` — and the code continued as if nothing happened. Tokens pulled from LPs sat trapped in the controller forever, invisible to everyone.

Other findings:

- **VolatilityInsurancePool** used a shared `claimedAmount` counter across multiple insurance events. Claiming from Event 1 reduced your payout from Event 2. LPs were being systematically shortchanged.
- **FederatedConsensus** checked vote expiry *after* state changes. The revert undid the `EXPIRED` status write, so proposals stayed `PENDING` forever — zombie proposals that could never be settled.
- **StablecoinFlowRegistry** had no bounds on flow ratios. A zero ratio caused division-by-zero downstream. Extreme ratios caused overflow.

Lesson: Silent failures are the most dangerous kind. If a call can fail, it must revert or be explicitly handled.

Pass 3: Governance & Infrastructure

The contracts that govern the contracts.

`DecentralizedTribunal` had two critical issues:

1. **Stake settlement used a push pattern.** If even one juror was a contract without a `receive()` function, `_settleStakes()` reverted for *everyone*. One bad actor could permanently block all tribunal settlements. We converted to a pull pattern with `pendingStakeWithdrawals` and a separate `withdrawStake()` function.

2. **No identity verification for jurors.** Anyone could `volunteerAsJuror()` — no soulbound identity, no level check, no reputation minimum. A sybil attacker could flood the juror pool with puppet accounts. We added `ISoulboundIdentityMinimal` integration with level and reputation thresholds.

We also verified UUPS upgrade safety across all 25 contracts — every single `_authorizeUpgrade` is properly gated with `onlyOwner`.

Lesson: Pull over push for ETH transfers. Always. One reverting recipient should never block a system.

Pass 4: Peripheral Contract Sweep

13 contracts. 7 fixes. The long tail.

This was the broadest pass — reviewing every contract outside the core flow:

- **WalletRecovery** used Solidity's `.transfer()` for ETH in three places (bond returns, slash payments, reporter rewards). Since `.transfer()` forwards only 2300 gas, any recipient with a non-trivial `receive()` function would fail. We replaced all three with `.call{value:}()` with success checks.

- **DisputeResolver** trapped excess ETH. If you sent 1.5 ETH to `registerArbitrator()` but the minimum stake was 1 ETH, the extra 0.5 ETH was permanently locked. Same issue in `escalateToTribunal()`. Added refund logic to both.

- **SoulboundIdentity** allowed instant recovery contract changes. If an owner's key was compromised, the attacker could immediately `setRecoveryContract()` and steal all identities. We replaced this with a 2-step timelock: `queueRecoveryContract()` then a 2-day wait then `executeRecoveryContractChange()`.

- **QuantumVault** emitted an event when keys were exhausted but didn't actually prevent reuse. The `AllKeysExhausted` error existed in the error declarations but was never triggered. Added the revert check.

Lesson: Peripheral contracts get less attention but the same attack surface. Audit everything.

Pass 5: Medium-Severity Hardening

Death by a thousand cuts.

Three fixes that prevent gas exhaustion and array manipulation:

- **DisputeResolver** had an unbounded arbitrator assignment loop. If all arbitrators were suspended, the `while` loop would iterate forever, consuming all gas. We bounded it by `activeArbitrators.length` with a hard `require` at the end.
- **ClawbackRegistry** had no limit on `caseWallets[caseId]`. An attacker could push addresses until any function iterating the array hit the block gas limit. Added `MAX_CASE_WALLETS = 1000`.
- **WalletRecovery** allowed setting `guardianThreshold` higher than the actual number of active guardians — making recovery permanently impossible. Added validation against `getActiveGuardianCount()`.

Lesson: Unbounded arrays and loops are ticking time bombs. Cap everything.

Pass 6: Compliance & Economic Safety

The rules that keep regulators happy.

- **ComplianceRegistry** checked KYC expiry in `canTrade()` but NOT in `canProvideLiquidity()` or `canUsePriorityAuction()`. A user whose KYC expired could still provide liquidity and access priority auctions — a clear compliance violation. Added expiry checks to both functions.
- **LoyaltyRewardsManager** calculated reward shares based on the `amount` parameter rather than actual tokens received. If a fee-on-transfer token was ever used as the reward token, `rewardPerShareAccumulated` would be inflated, paying out more than the contract actually holds. Added balance-before/after pattern.
- **SlippageGuaranteeFund** had a config field `userDailyLimitBps: 500` (5% of reserves) that was completely ignored. The actual daily limit was hardcoded to `1e18`. Dead config code. We wired it up properly.

Lesson: View functions need the same validation as state-changing functions. Dead config code means untested assumptions.

Pass 7: Cross-Contract Interactions

The final boss — composability risk.

The last pass examined how contracts call each other. Most concerns turned out to be false positives once you understand EVM atomicity:

- "Partial settlement leaves tokens stranded" — **False.** `settleBatch()` is a single atomic transaction. Tokens either fully settle or fully revert.
- "Admin can pause mid-swap" — **False.** A transaction is atomic. You can't pause in the middle of someone else's transaction.
- "Shapley tokens stuck if computation reverts" — **False.** `safeTransfer` + `computeShapleyValues` happen in the same TX. If computation reverts, the transfer reverts too.

But we did find one real issue:

DAOTreasury passed `0, 0` for slippage protection on both `addLiquidity` and `removeLiquidity`. This means an attacker monitoring the mempool could sandwich the treasury's liquidity operations — manipulating pool prices right before the treasury TX, then extracting the difference after. We added 95% minimum slippage protection on additions and caller-specified minimums on removals.

Lesson: Most cross-contract "vulnerabilities" dissolve once you understand EVM atomicity. But zero-slippage operations are always a real risk.

By the Numbers

Metric	Value
Audit passes	7
Contracts reviewed	25+
Vulnerabilities found	35+
Critical fixes	6
High fixes	14
Medium fixes	15+
False positives filtered	20+
New adversarial tests written	30+
Final test count	550
Test failures	0
Commits	14

Severity Breakdown

Critical (6): Silent token loss in IncentiveController, tribunal settlement DoS, tribunal sybil attacks, deploy script revert, cross-chain security bypass, rate limit bypass.

High (14): Insurance double-claim, vote expiry corruption, ETH trapped in disputes, unsafe `.transfer()`, instant identity theft, zero-slippage treasury, KYC expiry gaps, non-functional clawback, key exhaustion bypass, fund lock on zero-address.

Medium (15+): Deploy misconfigurations, unbounded loops, array growth, threshold validation, dead config code, fee-on-transfer risk, ratio bounds, quorum minimums.

What We Didn't Find

Equally important — what held up under scrutiny:

- **AMM core math** — constant product invariant, fee accounting, LP share calculations all verified through fuzz testing and formal invariant checks.
- **Commit-reveal mechanism** — hash hiding, slash accounting, priority bid ordering all battle-tested with adversarial tests.

- **TWAP oracle** — deviation checks, flash loan protection, same-block interaction blocks all functioning correctly.
- **Shapley distribution** — game theory axioms (efficiency, symmetry, null player, additivity) all formally verified.
- **DeterministicShuffle** — production code correctly uses `generateSeedSecure()` with block entropy, not the simplified `generateSeed()`.

The Meta-Lesson

Security isn't a single event. It's iterative.

Pass 1 found issues that Pass 7 wouldn't have caught. Pass 7 required understanding gained from Passes 1-6. Each layer of review builds context that makes the next layer more effective.

If you're building a DeFi protocol:

1. **Audit your deploy scripts.** Seriously.
2. **Silent failures kill.** Every external call needs explicit success handling.
3. **Pull over push** for ETH transfers. Always.
4. **Cap your arrays.** Unbounded data structures are gas bombs.
5. **View functions need validation too.** Access control gaps in read functions become compliance violations.
6. **Understand EVM atomicity** before panicking about cross-contract risks.
7. **Seven passes beats one.** Each round catches what the last round missed.

VibeSwap is an open-source omnichain DEX. The full codebase, including all security fixes and 550 tests, is available on GitHub.

Built in a cave. With a box of scraps.