SECURITY AUDIT

VolumeDynamicFeeHook

v2.4.0 · April 13, 2026

OBJECT	VolumeDynamicFeeHook (src/VolumeDynamicFeeHook.sol, 2 097 lines)
VERSION	v2.4.0
DATE	April 13, 2026
STATUS	Final
REPOSITORY	github.com/Axel-DeFi/VolumeDynamicFeeHook
AUDIT	Completed · Critical: 0 / High: 0 / Medium: 0 / Low: 0
AUDIT DEPTH	Deep
RUNTIME	EVM L1 / L2, Solidity ^0.8.26
HOOK ADDRESS	`0x2C3254Da...5044`
POOL ADDRESS	`0x226d6297...974c`

1. AUDIT ENVIRONMENT

Component	Version / Identifier
Model	claude-opus-4-6 · (reasoning: max)
Foundry (forge)	1.5.1
Solidity (solc)	0.8.26
Slither	0.11.5
Echidna	2.3.2
Halmos	0.3.3

Audited revision: `53147aae` (branch `main`, clean working tree).

2. EXECUTIVE SUMMARY

A full deep security audit of the `VolumeDynamicFeeHook` contract version v2.4.0 at revision `53147aae` has been completed. All mandatory deep audit checks were performed: manual code review (2 097 lines), full Foundry test suite (201 tests including invariant campaigns and fuzzing), Slither static analysis with manual triage of all findings, Echidna property testing (5 properties), and Halmos symbolic verification (16 properties).

No confirmed vulnerabilities of Critical, High, Medium, or Low severity were found. Five informational observations were recorded, relating to architectural decisions and external assumptions. Configuration-dependent risks are described separately and fall under operator responsibility.

The contract demonstrates a mature security architecture: strict parameter validation in the constructor and administrative functions, packed state without field overlap, limited external call surface, a deterministic state machine with confirmations and hold protection, a 48-hour timelock for hook fee percentage changes, two-step ownership transfer, and an emergency reset with immediate effect.

3. AUDIT OBJECT AND SCOPE

The `VolumeDynamicFeeHook` contract is a hook for Uniswap v4 pools with dynamic fees that tracks trading volume and automatically switches the LP fee rate between three modes:

- `FLOOR` (mode 0) — minimum fee during low volume;
- `CASH` (mode 1) — standard fee during normal volume;
- `EXTREME` (mode 2) — maximum fee during high volume.

Mode transitions are governed by a finite state machine that requires accumulating confirming periods and a hold protection mechanism that prevents premature rollback after an upgrade. Emergency reset (`emergencyReset`) allows the owner to force the contract into `FLOOR` or `CASH` mode, bypassing hold protection.

In addition to pool fee switching, the contract charges its own surcharge (Hook Fee) — a percentage of the estimated LP fee, accrued as an ERC6909 claim in `PoolManager`. The owner can withdraw accumulated funds via the `claimHookFees` function.

The audit scope includes:

- core hook logic: swap processing, fee calculation and accrual, volume tracking;
- mode transition state machine, including catch-up closing of overdue periods, idle reset, and low-volume reset;
- administrative functions: parameter management, timelock, ownership transfer, pause, and emergency reset;
- fund withdrawal: step-wise withdrawal of ERC6909 claims through the `PoolManager` accounting system;
- packing and unpacking of state into a single `uint256` slot;
- adjacent surfaces that could affect the protected properties.

4. PROTECTED PROPERTIES AND MATERIALITY LEVELS

The audit evaluates the contract against the following protected properties, ordered by priority for stakeholders.

Priority	Protected Property	Materiality
1	Preservation of LP funds and protection against unauthorized asset movement	PRIMARY
2	Privilege integrity: control over fees, modes, and state belongs only to authorized subjects	PRIMARY
3	Fee policy integrity: mode transitions, fee boundaries, and update rules conform to the declared model	PRIMARY
4	Pool liveness and recovery safety	PRIMARY
5	Accounting integrity of the hook as a system	SECONDARY
6	Owner revenue withdrawal path	ADMINISTRATIVE
7	Monitoring clarity, documentation, and operational ergonomics	CONTEXTUAL

Each observation and risk in this report is accompanied by the impact domain and materiality level. Informational observations and administrative risks are not placed on the same narrative level as risks affecting LP fund preservation or privilege integrity.

5. THREAT MODEL AND TRUST BOUNDARIES

The audit proceeds from three classes of adversaries and one environmental assumption.

Adversary Class	Capabilities
External user	Calling available external functions, supplying arbitrary input data, repeated calls, attempts to break invariants and cause denial of service
MEV adversary	Transaction reordering, exploiting execution order and transient states, attempting fee manipulation and forced mode transitions
Privileged owner	Executing all intended privileged actions, changing parameters within allowed bounds, potentially driving the system into an unsafe configuration if constraints are insufficient

Hostile environment assumption: external contracts, tokens, callbacks, and values originating from outside are treated as potentially hostile.

The contract's trust boundaries are established as follows.

Dependency	Trust Level	Rationale
`PoolManager` (Uniswap v4)	Trusted	Calls to `afterSwap`, `updateDynamicLPFee`, `mint`, `burn`, `take` are directed only to `poolManager`, whose address is set in the constructor and is immutable
Owner (`_owner`)	Trusted with constraints	Can change parameters only within hard-coded bounds; changing `hookFeePercent` requires a 48-hour timelock; cannot extract LP funds; cannot bypass `MAX_HOOK_FEE_PERCENT`, thresholds, or confirmation ranges
Swap input data	Untrusted	Validated: pool key, volume (dust threshold filter), swap result
`block.timestamp`	Environmental assumption	Dependency on block timestamp for period accounting, timelock, and idle detection; validator manipulation does not create critical impact due to period durations (hours-days)
Pool tokens	Trusted with constraints	Standard ERC20 behavior assumed; non-standard tokens (fee-on-transfer, rebase) may break volume accounting

6. ATTACK SURFACE MAP

Surface	Affected Property	Actor	Violation Type	Materiality
`afterSwap` — swap processing entry point	Fee policy, volume accounting, hook fee	External user (via `PoolManager`)	Volume manipulation, forced transitions	`PRIMARY`
`setModeFees` — setting fee rates	Fee policy	Owner	Incorrect rates (constrained by `floor` < `cash` < `extreme` validation)	`PRIMARY`
`setControllerSettings` — thresholds, confirmations, hold	Fee policy, liveness	Owner	Impossible or too-easy transitions	`PRIMARY`
`scheduleHookFeeChange` / `applyHookFeeChange` — timelock	Fund preservation, privileges	Owner	Timelock bypass, excessive surcharge (`MAX_HOOK_FEE_PERCENT` = 10)	`PRIMARY`
`emergencyReset` — emergency reset (pause-only)	Liveness, fee policy	Owner	Forced transition bypassing hold	`PRIMARY`
`pause` / `unpause` — pause	Liveness	Owner	Blocking swap processing (pool continues to operate)	`PRIMARY`
`proposeNewOwner` / `acceptOwnership` — transfer	Privileges	Owner, proposed owner	Privilege compromise upon key loss	`PRIMARY`
`claimHookFees` — hook fee withdrawal	Owner revenue	Owner	Withdrawal limited to accumulated amount; LP funds unaffected	`ADMINISTRATIVE`
`_withdrawCurrencyClaim` — step-wise withdrawal	Owner revenue	Owner (via `claimHookFees`)	Loop bounded by `int128` — not exploitable externally	`ADMINISTRATIVE`
`rescueToken` / `rescueETH` — asset rescue	Ergonomics	Owner	Access only to contract balance, not to ERC6909 claims in `PoolManager`	`CONTEXTUAL`
Packed state `_state` (`uint256`, 9 fields, 248 bits)	Accounting integrity, transitions	Internal (`_packState` / `_unpackState`)	Bit field overlap, data loss during packing	`PRIMARY`

7. METHODOLOGY AND ACTUAL COVERAGE

The audit was performed in five stages.

#	Stage	Method	Coverage
1	Manual code review	Sequential reading of all 2 097 lines of the contract, analysis of specification `docs/SPEC.md` (386 lines), invariant tests (406 lines), fuzz tests (241 lines), symbolic tests (815 lines), integration tests for claim accounting	Full contract including constructor, swap processing, transition state machine, administrative functions, state packing, fee calculation, fund withdrawal
2	Forge test suite	201 tests in 21 suites: 26 configuration and boundary tests, 61 administrative tests, 7 accounting integration tests, 28 invariant tests (4 configurations × 7 invariants × 128 runs × 8 192 calls), 4 fuzz tests (256 runs each), library helper tests	All major surfaces: mode invariants, fee boundaries, hook fee accounting, state packing, timelock, hold protection, configuration
3	Slither	Static analysis focused on the main contract (`--exclude-dependencies`); manual triage of all findings	42 medium-severity findings, 41 low, 4 informational — all analyzed and classified
4	Echidna	5 properties, 50 197 calls, property testing mode	Pack/unpack invariants, `feeIdx` bounds, EMA overflow, volume saturation, mode transition correctness
5	Halmos	16 symbolic properties, 842 paths, no solver timeout limit	Pack/unpack identity, all packed field bounds, fee ordering, arithmetic saturation, emergency reset, hold protection, transition confirmations, idle reset, clamp hold

8. AUTOMATED ANALYSIS RESULTS

Forge (build and full test suite)

Build completed without warnings (solc 0.8.26, 112 files). All 201 tests passed, 0 failed, 0 skipped. Invariant campaigns executed across four configurations (two stablecoins × two `tickSpacing` values), each with 128 runs × 8 192 handler calls with 11 operations — no violations or failures.

Test Type	Count	Result
Unit and configuration	128	128 / 128 passed
Administrative (including 2 fuzz tests)	61	61 / 61 passed
Invariant (4 configurations × 7 properties)	28	28 / 28 passed
Long-sequence fuzz tests	2	2 / 2 passed (256 runs)
Integration (claim accounting)	7	7 / 7 passed

Verified invariants include: `feeIdx` always within allowed bounds, strict ordering of modal fees, all packed fields within bit masks, timelock consistency, active hold matches mode settings, current LP fee matches current mode, hook fee accounting correctness at every step.

Slither (static analysis)

Executed on `src/VolumeDynamicFeeHook.sol` with dependency exclusion. Result: 0 high-severity findings, 42 medium, 41 low, 4 informational.

All 42 medium-severity findings were manually analyzed and classified as false positives or acceptable behavior. Main categories of false positives:

Category	Count	Reason for False Positive
Comparison of `enum` values with constants	~15	Intentional use of integer constants `MODE_FLOOR`, `MODE_CASH`, `MODE_EXTREME` instead of `enum` type — architectural decision for compatibility with state packing
Division before multiplication	~10	Intentional rounding down in `_hookFeeAmount` — in favor of LP, documented in specification
Reentrancy warnings through `poolManager`	~8	All external calls to `poolManager` occur within the managed `unlock` context; `poolManager` is a trusted dependency
Other (unused returns, shadowing)	~9	Intentional architectural decisions posing no security risk

No finding indicates an actual vulnerability or unaccounted-for pattern.

Echidna (property testing)

All 5 properties passed in 50 197 calls, no counterexamples found.

Property	Result
State pack/unpack identity	Passed
`feeIdx` always within bounds after arbitrary transitions	Passed
EMA does not exceed `uint96.max`	Passed
Volume saturates at `uint64.max`	Passed
Mode transitions conform to state machine rules	Passed

Halmos (symbolic verification)

All 16 symbolic checks passed, 842 symbolic paths explored, no counterexamples found.

Property	Paths	Result
`check_packUnpackRoundtrip` — pack/unpack identity	11	Passed
`check_feeIdxAlwaysBounded` — `feeIdx` $\in \{0, 1, 2\}$	320	Passed
`check_streakCountersNeverExceedBitWidth` — streak/hold counters within bit masks	320	Passed
`check_feeOrderingPreserved` — strict ordering `floorFee` $<$ `cashFee` $<$ `extremeFee`	4	Passed
`check_emaUpdateSaturatesAt96bit` — EMA saturation at `uint96.max`	8	Passed
`check_volumeAdditionSaturatesAt64bit` — volume saturation at `uint64.max`	4	Passed
`check_hookFeeNonNegativeAndClamped` — hook fee ≥ 0 and $\leq$ `int128.max`	9	Passed
`check_emergencyFloorPreemptsHold` — emergency reset ignores hold	10	Passed
`check_modeFeeSelectorConsistency` — each mode returns its own rate	7	Passed
`check_holdBlocksOrdinaryCashToFloor` — hold blocks CASH → FLOOR	13	Passed
`check_holdBlocksOrdinaryExtremeToCash` — hold blocks EXTREME → CASH	10	Passed
`check_extremeCanReachFloorOnlyViaEmergency` — EXTREME → FLOOR only via emergency reset	85	Passed
`check_idleResetClearsRuntimeState` — idle reset clears all counters	4	Passed
`check_cashToFloorNeedsFullConfirms` — CASH → FLOOR requires full confirmations	12	Passed
`check_extremeToCashNeedsFullConfirms` — EXTREME → CASH requires full confirmations	10	Passed
`check_controllerSettingsClampActiveHold` — hold clamped on settings change	10	Passed

Symbolic verification covers all critical arithmetic properties (overflow, saturation, bounds), state machine invariants (hold protection, confirmations, emergency reset), and state packing correctness.

9. COVERAGE GAPS AND QUESTIONS REQUIRING VERIFICATION

Area	Description	Impact on Audit Conclusions
Non-standard tokens	The contract assumes standard ERC20 behavior. Tests use standard mock tokens. Behavior with fee-on-transfer or rebase tokens has not been verified.	Not a contract vulnerability — this is a deployment assumption. The pool is bound to a specific pair at creation.
Economic modeling of MEV scenarios	Fuzzing and invariant tests verify transition correctness and accounting, but do not model a strategic MEV adversary optimizing profit through swap sequences.	Contract protected properties are not violated under MEV: fee boundaries and transitions are respected. Economic optimality of parameters is operator responsibility.
Behavior during prolonged pool inactivity	The scenario where `idleResetSeconds` expires across many periods followed by a sudden activity spike is covered at the state machine level (idle reset clears state), but not as a full economic scenario.	The state machine is deterministic; idle reset correctly clears all counters (confirmed symbolically).

All tools from the mandatory deep audit checklist were executed without omissions or failures. No blocking factors were identified.

10. FINDINGS TABLE

No confirmed vulnerabilities of Critical, High, Medium, or Low severity were found.

11. CONFIRMED FINDING CARDS

No confirmed vulnerabilities were found. This section is empty.

12. INFORMATIONAL OBSERVATIONS

ID	Title	Impact Domain	Materiality
I-01	Rounding down in hook fee calculation	`FEE_POLICY_INTEGRITY`	`CONTEXTUAL`
I-02	Catch-up closing semantics for overdue periods	`FEE_POLICY_INTEGRITY`	`CONTEXTUAL`
I-03	External assumption about owner key security	`PRIVILEGE_INTEGRITY`	`CONTEXTUAL`
I-04	Residual economic risk of volume manipulation	`FEE_POLICY_INTEGRITY`	`CONTEXTUAL`
I-05	Step-wise fund withdrawal with `int128` bound	`OWNER_REVENUE_PATH`	`ADMINISTRATIVE`

I-01. Rounding down in hook fee calculation. The `_hookFeeAmount` function (line 1641) performs two sequential integer divisions: first, the LP fee estimate is computed as `absUnspecified * appliedFeeBips / FEE_SCALE`, then the owner's share is taken as `lpFeeAmount * hookFeePct / 100`. Each division truncates the fractional part, systematically undercharging the surcharge by a few wei in favor of the trader and LP. This is an intentional architectural decision documented in the specification: the hook must not overcharge its share. Protected properties are not violated.

I-02. Catch-up closing semantics for overdue periods. When more than one period has elapsed since the last swap, the `_closePeriodsIfNeeded` function sequentially closes all overdue periods. The first closed period receives the entire volume of the current swap, while subsequent periods are closed with zero volume. The loop is bounded by the idle reset mechanism: if the pause exceeds `idleResetSeconds`, the state is cleared in a single operation without iteration. At the maximum ratio of `idleResetSeconds / periodSeconds` (constrained by the constant `MAX_IDLE_PERIODS` = 23), the loop performs no more than 23 iterations. This is a documented

architectural compromise (SPEC.md, section “Overdue Catch-Up”), accepted for simplicity and determinism. Protected properties are not violated.

I-03. External assumption about owner key security. The owner can change all configurable parameters within hard-coded bounds, pause swap processing, perform emergency resets, and withdraw accumulated hook fees. All privileged actions are constrained: `hookFeePercent ≤ 10` with a 48-hour timelock, modal fees undergo cross-validation, controller parameters are bounded by maximum constants. If the key is compromised, the adversary can set maximally unfavorable (but bounded) parameters and withdraw only accumulated hook fees — no access to LP funds. Two-step ownership transfer prevents accidental loss of control. Key security is an external responsibility.

I-04. Residual economic risk of volume manipulation. Trading volume is tracked on the stable side of the pool without an external price oracle. An adversary with sufficient capital can inflate volume through wash trading (offsetting swaps) to accelerate the transition to `EXTREME` mode. The impact is constrained by design: fee boundaries are set by the owner, transitions require accumulating confirmations across multiple periods, hold protection prevents immediate rollback, and the dust filter (`dustSwapThreshold`) rejects small swaps. The cost of the attack includes LP fees on each fabricated swap, making it economically unprofitable for most realistic configurations. This is a documented residual risk (SPEC.md).

I-05. Step-wise fund withdrawal with `int128` bound. The `_withdrawCurrencyClaim` function (line 1540) withdraws ERC6909 claims from `PoolManager` in chunks of at most `int128.max`, because the `PoolManager` accounting system operates with signed 128-bit integers. For realistic accumulated fee volumes, the loop performs exactly one iteration. The theoretical maximum number of iterations is `ceil(amount / int128.max)`, which for `uint256.max` is 2 iterations. The function is accessible only through `claimHookFees`, callable exclusively by the owner. External exploitation is impossible.

13. CONFIGURATION-DEPENDENT RISKS

The conditions listed below are not contract vulnerabilities. They describe configuration areas where the operator can create suboptimal or undesirable behavior within the allowed parameter values.

ID	Condition	Impact	Recommended Boundary
CS-01	Too small `periodSeconds` value	Increases the number of catch-up closing iterations during inactivity, raises gas cost of the first swap after a pause	Use values of 3 600 seconds (1 hour) or higher
CS-02	Close modal fee values (`cashFee ≈ extremeFee`)	Narrows the useful range of the dynamic strategy, making transitions economically insignificant	Ensure a meaningful gap between levels
CS-03	Small `idleResetSeconds` value	Frequent reset of accumulated controller state during natural trading pauses	Set at least several periods of inactivity
CS-04	Maximum `hookFeePercent` (10) with high modal fees	Surcharge up to 10% of estimated LP fee in `EXTREME` mode may be significant for traders	Calibrate `hookFeePercent` considering absolute fee rates

All listed parameters are validated by the contract at the time of setting. Combinations that lead to impossible transitions or fee ordering violations are rejected at the code level.

14. RESIDUAL RISKS

Risk	Description	Mitigating Factor
Owner key compromise	Adversary gains access to all privileged actions within the hard-coded constraints of the contract	Maximum damage is bounded: no access to LP funds, `hookFeePercent ≤ 10` with timelock, withdrawal limited to accumulated hook fees

Risk	Description	Mitigating Factor
Volume manipulation (wash trading)	Artificial volume inflation to accelerate transition to a higher mode	Cost of attack (LP fees), requirement for confirming periods, hold protection, dust filter
`PoolManager` v4 correctness	The contract trusts calls and accounting from `PoolManager`	External assumption; `PoolManager` is an audited protocol component of Uniswap v4
Non-standard tokens	Tokens with transfer fees or rebasing may break volume accounting accuracy	Deployment assumption; token pair is fixed in the constructor
Block timestamp accuracy	Validators can slightly shift `block.timestamp`	Periods are measured in hours-days; deviations of seconds do not affect protected properties

15. FINAL VERDICT

The `VolumeDynamicFeeHook` contract version v2.4.0 at revision `53147aae` has passed a full deep security audit with all mandatory checks.

No confirmed vulnerabilities of Critical, High, Medium, or Low severity were found. Five informational observations relate to documented architectural decisions and external assumptions, none of which violate the protected properties.

The contract demonstrates a consistent defensive architecture: strict validation of all parameters at setting and in the constructor, a deterministic transition state machine with confirmation requirements, packed state without field overlap (confirmed symbolically), constrained owner privileges with timelock, two-step ownership transfer, correct order of operations during fund withdrawal (internal accounting update first, then external call), and saturating arithmetic for EMA and volume.

The test suite covers all critical surfaces: 201 Foundry tests (including 28 invariant campaigns with 4 194 304 handler calls in total), 5 Echidna properties (50 197 calls), 16 Halmos symbolic checks (842 paths). Slither static analysis revealed no real security findings.

Residual risks are limited to external assumptions (owner key security, `PoolManager` correctness, token standardness) and residual economic impact (wash trading), which is constrained by the contract's design and the cost of attack.