INDEPENDENT SECURITY REVIEW

STILL
Smart Contract Security Audit Report

Final Report v1 — 31 August 2026

Project Name	STILL
Project Type	DeFi — ERC-4626 yield vault & fee harvester
Chain	Robinhood Chain (chain id 4663, EVM)
Compiler Version	Solidity 0.8.26 (pragma ^0.8.24), optimizer on, 1000 runs, evm_version = cancun
Repository	https://github.com/bretdevs/still
Commit Under Review	6b1ae8179073a6aee2fa88f3eec68334762f4211 (branch main)
Deployment Status	Not deployed at time of review
Reviewer	Independent security review

Table of Contents

Executive Summary

The STILL team commissioned an independent security review of their smart contracts. The reviewer manually and proactively reviewed the code in order to provide the project's team and community with an independent assessment of the security of the contracts prior to deployment.

The review covered the complete in-scope Solidity codebase (~292 nSLOC): an OpenZeppelin-based ERC-4626 vault (`StillVault`) and a permissionless fee harvester (`StillHarvester`) that converts trading-fee ETH into STILL and delivers it to the vault. No critical or high-severity vulnerabilities were identified. One medium-severity economic issue, two low-severity issues, and four quality-assurance items are reported below.

Project Context

STILL is an ERC-4626 vault on Robinhood Chain (chain id 4663, EVM) for a token launched through the pons v2 launchpad. The share token vSTILL is a plain ERC-20. The token's trading fees (the pons standard fee share plus a 3% creator tax) accrue as ETH in the pons fee escrow. A permissionless harvester claims that ETH, buys STILL (on the pons bonding curve before graduation, on the Uniswap v4 pool after) and transfers the STILL to the vault. No shares are minted for harvested tokens, so STILL per vSTILL only moves one way. Neither contract has an owner, admin, pause, upgrade path or parameter setter.

Property	Value
Project Name	STILL
Project Type	DeFi
Compiler Version	^0.8.24 (built with 0.8.26)
Chain	Robinhood Chain (4663)
Website	https://holdstill-plum.vercel.app (demo mode at time of review)
Language	Solidity

Audit Scope

The review was performed on the source code snapshot at commit `6b1ae8179073a6aee2fa88f3eec68334762f4211` of the public repository github.com/bretdevs/still.

File	nSLOC	Description
`src/StillVault.sol`	14	OpenZeppelin ERC4626 plus ERC20Permit over STILL. Overrides `decimals()` only.
`src/StillHarvester.sol`	165	Creator fee recipient. Permissionless `harvest(minStillOut)`; escrow claim and fee sweeps; curve buy; Uniswap v4 pool buy via `unlockCallback`; small on-chain ledger.
`src/interfaces/IPons.sol`	85	The slice of pons v2 (factory, escrow, curve, hook) the harvester touches.
`src/interfaces/IUniswapV4.sol`	28	PoolManager, PoolKey and IUnlockCallback slice.

Included as context (findings welcome but secondary): `script/Deploy.s.sol`, `test/` (30 unit tests plus 8 fork tests against Robinhood Chain), and `test/mocks/Mocks.sol`.

Out of scope: the pons v2 protocol contracts (factory, escrow, curve, hook) and the Uniswap v4 PoolManager, treated as trusted external dependencies; OpenZeppelin Contracts v5.4.0, of which only the composition is reviewed; and the off-chain keeper, website and brand assets.

Security Rating

> **Overall rating: Standard risks identified.** One medium-severity economic issue (MEV / griefing exposure on fee buys), two low-severity issues, and four quality-assurance items. No critical or high-severity vulnerability was identified, and no fund-extraction (rug/drain) path exists in the reviewed code. The medium issue is bounded by immutable protocol parameters and can be mitigated or disclosed at the team's discretion.

Severity	Count	Status
Critical	0	—
High	0	—
Medium	1	Open
Low	2	1 Open, 1 Acknowledged
QA	4	Open

Intended Smart Contract Functions

Contract	Function	Intended behaviour
StillVault	`deposit / mint / withdraw / redeem`	Standard ERC-4626 vault over STILL; rounding favours the vault; no fees, no hooks.
StillVault	`decimals()`	Returns the underlying asset decimals (18) via correctly-specified multiple-inheritance override.
StillHarvester	`harvest(minStillOut)`	Permissionless. Sweeps and claims fee ETH from pons, buys STILL (curve pre-graduation, Uniswap v4 pool post-graduation) capped by `maxBuyPerHarvest` and `cooldown`, delivers STILL to the vault, appends a ledger entry.
StillHarvester	`unlockCallback(data)`	PoolManager-only callback performing an exact-input ETH→STILL swap, enforcing `minStillOut`, settling ETH owed and taking STILL directly to the vault.
StillHarvester	`pending / canBuy / phase / poolKey / poolId / recent / ledgerLength`	View helpers for the keeper and the public website.
StillHarvester	`receive()`	Accepts escrow claims, curve refunds and unspent swap input.

Code Quality

The codebase is small, well-organised and extensively documented. The project ships 30 unit tests (21 harvester, 9 vault) and 8 fork tests against the live pons v2 stack; the package's own test run reports 30/30 unit tests passing at the reviewed commit, including a 512-run fuzz test enforcing the monotonic-ratio invariant and a dedicated inflation-attack test. NatSpec is present on all public surfaces, external dependencies are pinned as submodules, and the contract deliberately minimises surface area (no owner, no admin functions, no upgrade path).

Areas where the project could improve: the on-chain ledger grows unboundedly (QA-01); the deploy script's seed step — the sole inflation-attack defence — is optional (L-01); and constructor arguments for external dependencies are not cross-validated against the launch record (QA-03).

Audit Resources

This was a manual, line-by-line static security review assisted by automated analysis tooling pipeline. Every in-scope file was reviewed in full, including the flattened sources in `flat/`. The complete fee path (escrow → claim → harvest → curve buy / Uniswap v4 unlock → swap → settle → take → vault) was traced end-to-end, and the project's own unit and fork tests were analysed. The build and test suite were not re-executed in the review environment (no Foundry toolchain/submodules in the audit package); re-running them before deployment is recommended in the Conclusion.

Dependencies

Dependency	Version / Commit	Usage
OpenZeppelin Contracts	v5.4.0 (`c64a1edb`)	ERC20, ERC4626, ERC20Permit in StillVault; IERC20 in StillHarvester.
forge-std	v1.16.2 (`bf647bd6`)	Tests and deploy script only (not in runtime bytecode).
pons v2 (external)	Live on Robinhood Chain	Factory, fee escrow, bonding curve, meme hook. Trusted external dependency.
Uniswap v4 PoolManager (external)	Live on Robinhood Chain	unlock/swap/settle/take for post-graduation buys. Trusted external dependency.

Severity Definitions

Severity	Definition
Critical	Direct loss of user funds or full compromise of the system with no unrealistic preconditions. Requires immediate action before any deployment.
High	Significant loss of funds or permanent breakage of a core protocol invariant under plausible conditions.
Medium	Bounded loss of value, yield degradation, or griefing feasible under normal operation; or a material risk that is only partially mitigated by protocol parameters.
Low	Minor impact, or impact contingent on operator error, edge-case states, or specific external conditions.
QA	Quality assurance and best-practice items with no direct security impact: hardening, documentation, gas or code-clarity recommendations.

Status Definitions

Status	Definition
Resolved	The finding has been remediated in the reviewed code.
Acknowledged	The project team is aware of the finding and accepts the associated risk; it is publicly documented by the team.
Open	The finding is reported for the team's consideration; no remediation has been applied at the reviewed commit.

Audit Findings

[M-01] StillHarvester#harvest — Permissionless buys with caller-supplied slippage enable MEV / griefing

Medium Open

DESCRIPTION

`harvest(minStillOut)` is callable by anyone, and the slippage floor `minStillOut` is entirely caller-controlled. A hostile caller can invoke `harvest(0)`, executing a zero-slippage market buy of the accumulated fee ETH.

VULNERABILITY DETAILS

```
function harvest(uint256 minStillOut) external returns (uint256 ethSpent, uint256 stillBought) {
    ...
    if (held > 0 && canBuy()) {
        uint256 budget = held > maxBuyPerHarvest ? maxBuyPerHarvest : held;  // cap on size only
        uint8 p = phase();
        if (p == 0) {
            (ethSpent, stillBought) = _buyOnCurve(budget, minStillOut);  // minStillOut = 0 => no protection
        } else if (p == 2) {
            (ethSpent, stillBought) = _buyOnPool(budget, minStillOut);   // minStillOut = 0 => no protection
        }
        ...
    }
}
```

The slippage parameter protects only against the keeper's own quoting error; it offers no protection against a deliberate actor. Because the buy is a deterministic market order (bonding-curve buy pre-graduation, full-range exact-input swap post-graduation), a third party can front-run a harvest — or trigger their own harvest at a moment of their choosing — pushing the price up before the buy and unwinding after it.

IMPACT

Continuous, repeatable yield degradation: each manipulated buy returns fewer STILL for the same fee ETH, so the vault's share price accrues more slowly than the fees would otherwise justify. The cost is socialised across all vSTILL holders. The damage is bounded — per call by `maxBuyPerHarvest` (0.1 ETH default) and per unit time by `cooldown` (5 minutes default), a maximum displacement of ~28.8 ETH/day — and STILL still lands in the vault, so nothing is lost outright. On the bonding curve the attack economics are further degraded by the 1% curve fee and the 3% creator tax (which flows back to the harvester); on the zero-fee Uniswap v4 pool only the hook fee and gas apply, making post-graduation sandwiching the more plausible leg.

RECOMMENDATION

1. Keep the keeper quoting with tight slippage (`SLIPPAGE_BPS` , default 300 bps) — already the design — and publish the fact that `harvest(0)` is callable by anyone, safe only because of the cap.
2. Consider an in-contract floor on `minStillOut` (rejecting calls below a conservative bound). This materi-

ally raises griefing cost at the price of a small amount of additional protocol opinion.
3. Consider a wider `cooldown` to reduce the griefing rate, at the cost of slower fee capture.
4. If left unchanged, disclose the exposure in the project's published risk list.

STATUS

Open

[L-01] Deploy.s.sol#_seed — Inflation / first-depositor defence depends entirely on an optional deploy-time seed

Low Open

DESCRIPTION

`StillVault` is plain ERC-4626 whose only built-in inflation guard is OpenZeppelin's 1-wei virtual asset/share offset. The effective defence is the deploy script seeding the vault and parking the resulting vSTILL at `0x...dEaD` — and that step is optional.

VULNERABILITY DETAILS

```
function _seed(Cfg memory c, Out memory o) internal {
    if (c.seedEth == 0) return;   // <-- seed is optional
    uint256 got = IPonsCurve(o.curve).buy{value: c.seedEth}(c.seedEth, 0, c.deployer);
    uint256 toDeposit = c.seedStill > got ? got : c.seedStill;
    IERC20(o.token).approve(o.vault, toDeposit);
    StillVault(o.vault).deposit(toDeposit, DEAD);  // shares parked at dead address
    ...
}
```

The defence's effectiveness is proportional to the seeded share amount. If `SEED_ETH_WEI` / `SEED_STILL_WEI` are zero or negligible (or the manual fallback path in the README is used without the seed), `totalSupply` starts near zero and the classic ERC-4626 donation/inflation attack becomes profitable again.

IMPACT

With the documented defaults (0.05 ETH seed, 1,000 STILL parked at `0x...dEaD`) the attack is unprofitable and victim loss is dust, as demonstrated by `test_inflationAttackIsUnprofitableAfterSeed`. The risk is therefore conditional on operator error at deploy time, not on normal operation.

RECOMMENDATION

Make the seed mandatory: revert in `_seed` if `seedEth == 0` or if the seeded share value is below a minimum. After deployment, verify on-chain that vSTILL parked at `0x...dEaD` exceeds the minimum and that the ratio is near 1.0 before pointing the fee recipient at the harvester; the README's manual fallback path should include the same seed step.

STATUS

Open

[L-02] StillHarvester#harvest — Fee ETH permanently locked in swept (1) and rescued (3) phases

Low Acknowledged

DESCRIPTION

In phases 1 (swept) and 3 (rescued) there is no buy path; claimed ETH simply remains in the contract. Because the contract has no owner and no recovery function, that ETH can never be moved except by a later buy.

VULNERABILITY DETAILS

```
if (p == 0) {
    (ethSpent, stillBought) = _buyOnCurve(budget, minStillOut);
} else if (p == 2) {
    (ethSpent, stillBought) = _buyOnPool(budget, minStillOut);
}
// phase 1 (swept, pool not yet created) and 3 (rescued): nothing to buy from. ETH waits.
```

IMPACT

Phase 1 is transient (the pool is created shortly after graduation), so the impact there is negligible. Phase 3 (rescued) is terminal: fee ETH already claimed at the time of a rescue is permanently locked. This is already documented as an accepted risk in the project README and audit scope. The review confirms it is not worse than stated — no double counting and no loss of user deposits; only accumulated fee ETH becomes unrecoverable.

RECOMMENDATION

Optionally add a rescue-specific, permissionless path that forwards claimed ETH to the vault as a donation when `phase() == 3`, so a rescue does not strand value. Any such path must remain permissionless and revert-safe to preserve the no-owner guarantee. If unchanged, keep the item in the published risk list.

STATUS

Acknowledged — documented by the team as an accepted risk.

[Q-01] StillHarvester.Entry — Packed ledger fields can truncate under extreme parameters

QA Open

DESCRIPTION

```
struct Entry {
    uint40 blockNumber;
    uint40 timestamp;
    uint112 ethSpent;
    uint112 stillBought;
    uint112 ratioAfter;   // STILL per 1e18 vSTILL, after this harvest
}
```

`ratioAfter = uint112(ratio)` where `ratio = convertToAssets(1e18)`. With a realistic seed and the 1B-token supply this fits comfortably within `uint112` (> 5e33). Truncation would require a negligible seed combined with the full supply accruing to a tiny remaining supply — i.e. a misconfiguration already covered by L-01. Field truncation affects only the on-chain display ledger, never funds.

RECOMMENDATION

Use `uint256` for `ratioAfter` (or document an explicit bound) as defence-in-depth.

STATUS

Open

[Q-02] StillHarvester#unlockCallback — No cap/cooldown enforcement inside the callback

QA Open

DESCRIPTION

The per-call cap and cooldown are enforced only in `harvest()`'s budget calculation, not inside `unlockCallback`. The callback accepts its `budget` from caller-supplied `data` and settles whatever `owed` the swap reports. This is safe today because the callback is gated to `msg.sender == address(poolManager)`, `poolManager` is an immutable derived from the launch/hook at deploy time, and the callback is only reachable via `harvest()` → `_buyOnPool()`, where the budget is already capped by `min(held, maxBuyPerHarvest)`.

IMPACT

Hypothetical only: a compromised or misconfigured PoolManager could invoke `unlockCallback` directly with an uncapped budget, spending all held ETH. Flagged for awareness; no change required given the PoolManager is a trusted immutable external dependency.

RECOMMENDATION

Optionally re-derive or bound the budget inside the callback; otherwise document the trust assumption.

STATUS

Open

[Q-03] StillHarvester constructor — External dependency addresses not cross-verified against the launch record

QA Open

DESCRIPTION

The constructor validates the launch record (`UnknownLaunch`, `NativePairOnly`) and derives `curve`, `poolFee` and `tickSpacing` from it, but `escrow_`, `hook_` and `poolManager_` are accepted as-is from deployment arguments. A misconfigured `escrow` would leave fees credited to an escrow the harvester never claims from; a wrong `hook` or `poolManager` would produce a wrong `poolId()` /pool key. These are de-

ploy-time operator errors (self-inflicted, not third-party exploitable), and the deploy script already reads them from environment variables with an explicit re-check warning.

RECOMMENDATION

In `Deploy.s.sol`, derive `escrow` and `hook` from the pons factory where possible and assert `IPonsMemeHook(hook).poolManager() == poolManager` before broadcasting.

STATUS

Open

[Q-04] StillHarvester#recent — Caller-sized array allocation; ledger grows without bound

QA Open

DESCRIPTION

`recent(n)` allocates `new Entry[](n)` for caller-supplied `n` (capped only at the current ledger length), and `_ledger` grows by one 40-byte entry per successful harvest (~288/day at the default cooldown). The cost is borne by the caller (a view call), so there is no contract-level denial of service, but a huge `n` produces a large return payload and the unbounded storage growth accumulates over years. The site's `recent(12)` usage is unaffected.

RECOMMENDATION

Optionally cap `recent` to a sane maximum (e.g. 256) and/or stop appending after a maximum ledger length.

STATUS

Open

Centralisation

The STILL project values decentralisation and immutability over operator control.

Neither reviewed contract has an owner, admin role, pause, upgrade path, or parameter setter. All harvester bounds (`maxBuyPerHarvest` , `cooldown`) are immutable after deployment, and the fee destination is fixed by the pons factory record. The only residual centralisation vectors are external to the reviewed code: the pons v2 protocol can redirect creator fees through its CTO mechanism behind a three-day timelock, and post-graduation fee sweeps requiring an internal swap are operator-only on the pons side. Both are documented by the team as accepted risks.

Conclusion

After the reviewer's analysis, the STILL project presents a sound, minimal and well-tested codebase. The core security property — the harvester's ETH can only ever be converted into STILL that lands in the vault, never extracted — holds under review, and no fund-extraction path, reentrancy issue, or access-control surface was identified. The Uniswap v4 accounting in `unlockCallback` (BalanceDelta decoding, net-output slippage check, exact settle/take) is correct and consistent with the project's fork tests.

One medium-severity economic issue remains open (M-01): permissionless buys with caller-controlled slippage expose fee purchases to bounded MEV/griefing, mitigated but not eliminated by the immutable cap and cooldown. The team should either tighten this or disclose it publicly. The deploy-time seed (L-01) should be made mandatory before launch. All other items are low-severity or quality-assurance in nature.

To the best of our ability, we were not able to identify any further vulnerabilities. Before deployment the team should re-run the full suite, including the fork tests against the live pons v2 stack:

```
git clone --recurse-submodules https://github.com/bretdevs/still.git
cd still && git checkout 6b1ae8179073a6aee2fa88f3eec68334762f4211
forge build
forge test --no-match-path test/Fork.t.sol        # 30 unit tests
FORK=1 forge test --match-contract ForkTest -vv    # 8 fork tests (needs RPC)
```

Our Methodology

This review follows a transparent review process and treats audits as a collaborative effort. The objective of our security reviews is to improve the quality of the systems we examine and to aim for sufficient remediation to help protect users and project leaders. Below is the methodology used in this review.

Manual Code Review: In manually analysing all of the code, we seek to find any potential issues with code logic, error handling, accounting arithmetic, external-call handling, reentrancy, and access control. We also watch for areas where more defensive programming could reduce the risk of future mistakes. Although our primary focus is the in-scope code, we examine dependency code and behaviour where relevant to a line of investigation.

Vulnerability Analysis: Our methodology combines manual code analysis with threat modelling of the specific system: the permissionless harvest path, the ERC-4626 first-depositor/inflation surface, the Uniswap v4 flash-accounting cycle (`unlock` / `swap` / `settle` / `take`), slippage semantics across curve and pool buys, cap/cooldown griefing, phase-transition states, and claim-failure handling. We reviewed the project's specifications (README, audit scope), its unit and fork tests, and its mocks to validate the intended behaviour against the implementation.

Documenting Results: When a potential issue is discovered, an entry is created in this document even before feasibility and impact are fully verified; suspicions are then confirmed or dismissed through code analysis and, where possible, against the project's own test evidence. Confirmed issues are assigned a severity per the definitions above, with concrete impact and remediation guidance.

Suggested Solutions: We suggest mitigations applicable to the current design and remediation requirements for any future revision. Mitigation and remediation recommendations should be scrutinised by the developers; successful mitigation is an ongoing collaborative process after delivery of this report and before contract details are made public.

Disclaimers

Reviewer's Disclaimer. This review was performed in accordance with good industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in the smart contract source code disclosed in this report, and the source code's compilation, deployment and intended functionality. Due to the fact that the total number of test cases is unlimited, this review makes no statements or warranties on the security of the code. It cannot be considered a sufficient assessment of the utility and safety of the code, its bug-free status, or any other statement of the contract. You should not rely on this report alone; we additionally suggest a bug bounty program to confirm the ongoing security of the smart contracts. The reviewer is not responsible for the safety of any funds and is not in any way liable for the security of the project.

Attribution Disclaimer. This report is deliberately unattributed: it names no review firm or individual. It is **not** an audit by Hashlock Pty Ltd or any other named audit firm, and it must not be published, marketed, or otherwise represented as one. It may be described only as an “Independent smart contract security review”. Only a report issued by an audit firm itself may carry that firm's name.

Technical Disclaimer. Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to attacks. This review cannot guarantee the explicit security of the audited smart contracts. The pons v2 protocol contracts and the Uniswap v4 PoolManager were treated as trusted external dependencies and were not audited.