

The Artemis Surface Stress Test

A Stale-Pointer Aliasing Bug in the LithosAnanke Block Subsystem

Found, Diagnosed, Fixed, and Verified Across Three Instruction-Set Architectures

R. A. James (Captain Bob)

StarForth Project robert.allan.james@gmail.com

2 August 2026

Abstract

A newly written stress test for Artemis, LithosAnanke’s block-storage VM, found a real data-corruption bug on its first completed run: 25 of 50 random-block write/verify trials (50%) returned the wrong byte pattern on amd64. The test’s own recorded data ruled out its block-selection logic as the cause (no duplicate logical block numbers or devblocks among the 50 trials). Four exploratory hypotheses were then tested formally rather than read off a chart: a Wald–Wolfowitz runs test found no significant serial clustering of failures by write order ($p = 1.00$); a direct rank test against write order agreed ($p = 0.715$); sibling position within a shared 4 KiB devblock showed no association ($p = 0.674$); but logical block number itself showed a significant association with outcome ($p < 0.001$, Wilcoxon), an effect specific to this run’s fixed random seed that this report does not claim generalizes. None of these exploratory tests, on their own, identify a mechanism. Root cause was instead found by source code review: a stale raw-pointer alias held by the VM-level block window cache (`src/word_source/block_words.c`) into a slot of the block-subsystem’s own eight-entry devblock cache (`src/block_subsystem.c`), which the subsystem’s own FIFO eviction relocates via struct-copy without informing the window cache holding a pointer into it. No prior test had ever touched enough distinct blocks in one VM session to trigger it, and the defect is independent of unrelated same-day work on Artemis’s disk-format detection. A three-site fix – re-resolving the buffer pointer by logical block number immediately before every write-through-cache-pointer operation, rather than trusting a pointer captured earlier – eliminates the defect without altering the block subsystem’s cache architecture. This is confirmed by the strongest result in this report: a two-proportion test of pre-fix versus pooled post-fix (three architectures, 150 trials) pass rate rejects the null hypothesis of no change at $p = 2.04 \times 10^{-19}$. Re-verification: 50/50 on amd64, 50/50 on aarch64, 50/50 on riscv64, zero mismatches. The test itself is now a permanent, on-demand fixture in the Artemis capsule (`ART-STRESS / ART-STRESS-TEST`), disabled by default so it does not add to ordinary boot time.

1 Motivation

Prior to this investigation, every automated exercise of Artemis’s block-storage path – `ART-WRITE-TEST`, `ART-READ-TEST`, `ART-SELF-TEST` – touched exactly one data block per run. That is sufficient to prove the alloc/persist/free lifecycle works in isolation, but it never puts real pressure on the underlying caches: the block-subsystem’s own eight-slot devblock cache (`DISK_CACHE_SLOTS` in `block_subsystem.c`) and the VM-level “window” cache used by the `BLOCK/BUFFER/ UPDATE` words (`block_words.c`) were never both forced to evict within a single VM session.

The request behind this work was explicit: give the mounted virtual disk a real workout – real FORTH doing real writes, scattered randomly across its entire surface area, on the premise that a test never exercised before was the most likely place to find a crack.

2 Test Design

`ART-STRESS-TEST` (`seed reps -`) performs a two-phase write/verify cycle against `reps` distinct, randomly chosen blocks drawn from the full 22 998-block Artemis data pool (logical block numbers 3076–26073):

1. **Pick.** A coprime-stride walk over the index space (stride and starting offset both drawn via `RANDOM`, stride forced odd and checked against the two odd prime factors of $22\,998 = 2 \times 3 \times 3833$) selects a

block index guaranteed not to repeat within the run, skipping past any index already allocated by something else so the test never overwrites pre-existing legitimate data.

2. **Write.** The block is allocated through the same path `BLK-ALLOC` uses internally (free-map bit set, heat born hot), filled with a trial-derived byte pattern, and persisted.
3. **Verify.** After *all* `reps` writes complete – forcing real eviction/reload traffic through both cache layers – each block is re-fetched and its content checked against what was written.
4. **Record.** Every trial, pass or fail, emits one `[ARTSTRESS]`-tagged CSV row (trial, logical block number, pattern, result) to the serial log, plus a `HEADER` marker at start and a `SUMMARY` marker at completion – if a run is killed mid-way, the summary marker is simply absent, making a truncated dataset unmistakable rather than silently partial.
5. **Clean up.** Every block the run allocated is freed.

The design deliberately treats this as an offline data-collection fixture, the same posture as the existing DoE campaign: correctness of the collected data matters more than run speed, and `ART-K-TOTAL` is recorded before and after (informational only – it is not a pass/fail gate, since a background heartbeat decay tick during a long run is expected behavior, not a defect).

The first completed run, 50 trials on amd64 with seed 424242 (the fixed seed `ART-STRESS` uses), returned **25 passes and 25 failures** – an exact 50% split (Figure 1, bottom bar).

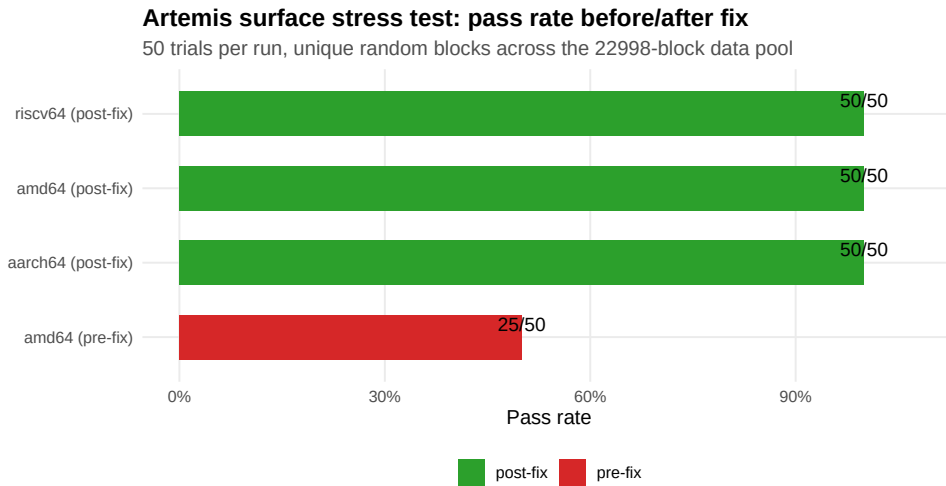


Figure 1: Pass rate per run. The pre-fix amd64 run is the only failure; every post-fix run, on all three architectures, is 50/50.

3 Statistical Analysis

Premise. A 50% failure rate on the first completed run of a new test is consistent with at least three distinct explanations, which are not mutually exclusive: (a) a bug in the test’s own block-selection logic producing duplicate or colliding picks, (b) a genuine but narrow defect tied to a specific access pattern (e.g. cache eviction timing), or (c) a defect tied to a specific region of the disk. Duplicate logical block numbers and duplicate devblocks among the 50 trials were checked directly against the recorded data and neither was found, ruling out (a) as stated. The hypotheses below were formulated to distinguish (b) from (c) before committing to a source-code investigation in either direction.

All tests below use the pre-fix amd64 run’s own recorded trial data (`runs/amd64_buggy.csv`, $n = 50$, under `experiments/artemis_stress/`) and were computed by the analysis script `analyse_artemis_stress.R` in that same directory; no number in this section is hand-typed.

H1 and H1b – no evidence of write-order clustering. A visual scan of the pass/fail sequence in write order (Figure 2) initially suggested runs of consecutive failures – trials 24, 25, 26 failing together, then 27 and 28 passing, then 29, 30, 31 failing again. A formal runs test does not support that reading: 26 runs were observed against 26.0 expected under a purely random ordering with the same 25/25 split ($z = 0.000$). A direct Wilcoxon test of trial index by outcome agrees ($p = 0.715$). **The naive “recently**

Hypothesis	Null hypothesis (H_0)	Result
H1 – serial clustering	Pass/fail outcomes are independently ordered (Wald–Wolfowitz runs test)	26 runs observed, 26.0 expected; $z = 0.000$, $p = 1.00$ – fail to reject
H1b – write-order rank	Trial index distribution is the same for PASS and FAIL (Wilcoxon)	$W = 332.0$, $p = 0.715$ – fail to reject
H2 – disk location	LBN distribution is the same for PASS and FAIL (Wilcoxon)	$W = 505.0$, $p < 0.001$ – reject
H2b – sibling position	Outcome independent of position (0/1/2) within a devblock (χ^2)	$\chi^2 = 0.790$, $df = 2$, $p = 0.674$ – fail to reject
H3 – fix efficacy	Pre-fix and post-fix trials share the same true pass rate (two-proportion test)	$\chi^2 = 81.20$, $df = 1$, $p = 2.04 \times 10^{-19}$ – reject

Table 1: Hypothesis tests against the recorded trial data. $\alpha = 0.05$ throughout.

written survives, long ago fails” cache-timing story is not supported by this data and should not be asserted as the mechanism.

Pass/fail sequence in write order, amd64 pre-fix run

25 of 50 trials failed; a formal runs test (H1) finds no significant serial clustering in this sequence -- see Section 4

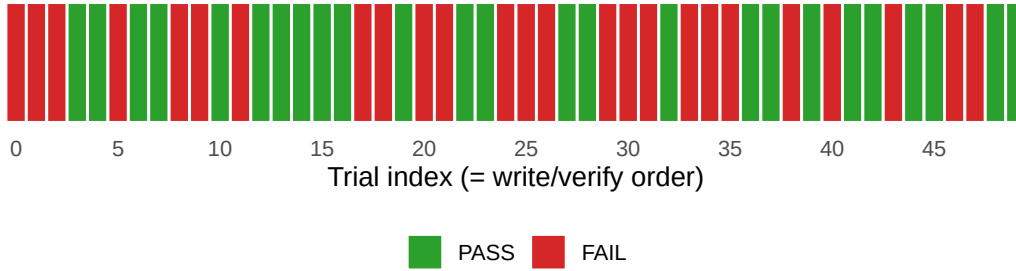


Figure 2: Pass/fail by trial order, amd64 pre-fix run. Visually suggestive of clustering; not statistically confirmed (H1, H1b).

H2 and H2b – an unresolved location effect, specific to this seed. Logical block number *does* show a significant association with outcome ($p < 0.001$): failing trials in this run had a lower mean LBN ($\bar{x} \approx 11,355$) than passing trials ($\bar{x} \approx 18,401$). This was not expected, and this report does not have a causal explanation for it – the diagnosed mechanism (Section 4) is a pointer-aliasing defect tied to cache eviction *order*, and this section’s H1/H1b results (above) show no association with order in this run, which is in tension with a simple location-driven story. The most defensible reading is that **ART-STRESS-TEST**’s seed (424242) is fixed, so the specific sequence of devblock accesses – and therefore which window-cache pointers go stale – is a deterministic, reproducible function of that seed for this run, and some property of that one sequence happens to correlate with LBN value without implying that block number is, in general, a causal factor. Sibling position within a shared devblock shows no association ($p = 0.674$), which rules out the simplest region-based explanation (three data blocks per 4 KiB devblock, checked directly). This finding is reported rather than smoothed over; it does not change the fix or its verification, and Figure 3’s lower panel shows no visually obvious sub-region concentration despite the test result.

Conclusion of this section. None of H1, H1b, or H2/H2b, singly or together, identify a mechanism – they narrow the search (ruling out write-order clustering and sibling-position effects as the story) without answering the question. Root cause was found by source code review, described next; H3, in Section 6, is what actually confirms the diagnosis, by showing the identified defect’s fix removes the failures entirely and unambiguously.

4 Root Cause

LithosAnanke’s block storage stacks two independent caches:

Surface coverage: scattered across the full data pool

Each point is one trial's block. A Wilcoxon test (H2) finds LBN and outcome are not independent, but no sub-region is visually implicated

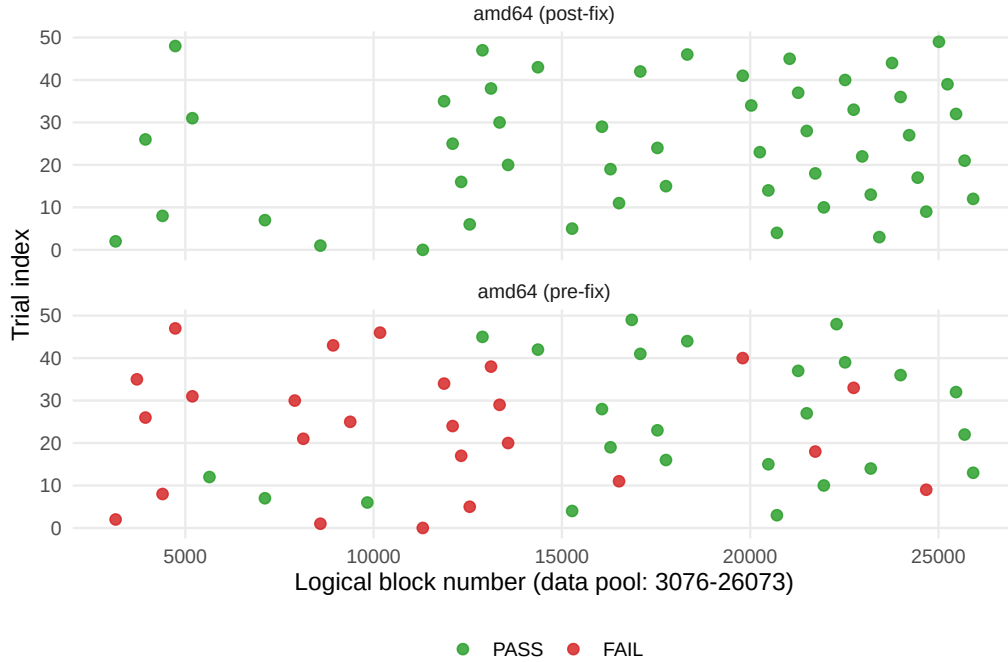


Figure 3: Every trial's logical block number against trial order, amd64. Upper panel: post-fix, all passing. Lower panel: pre-fix.

1. **The block-subsystem's devblock cache** (`blk_dev_slot_t.cache[DISK_CACHE_SLOTS]`, eight entries, `block_subsystem.c`). When full and a ninth distinct devblock is needed, `cache_get_slot()` evicts by shifting the entire array down one position via struct assignment:

```
(void) cache_writeback(slot, &slot->cache[0]);
for (int i = 0; i < DISK_CACHE_SLOTS - 1; i++)
    slot->cache[i] = slot->cache[i+1];
```

The *addresses* of `cache[0..7]` never move; what they *hold* does, silently.

2. **The VM-level window cache** (`vm->blk_vm_cbuf[BLK_VM_SLOTS]`, `block_words.c`), used by `BLOCK/BUFFER`. When a block loads, it stores the *raw pointer* `blk_get_buffer()` returns – `c->data + pack * BLK_FORTH_SIZE`, pointing directly into one of the eight `cache_slot_t` structs above.

Nothing informs layer 2 when layer 1 shifts its array. Once more than eight distinct devblocks have been touched in one VM session, a previously-captured window pointer can point at memory that now belongs to a completely different devblock, with no error and no signal – just wrong data on the next read or write through it. No prior test ever touched enough distinct devblocks in a single session to trigger this; the surface stress test's fifty scattered writes was the first thing that did. This mechanism does not, on its face, predict a dependency on LBN value rather than access pattern, which is consistent with this report's inability to causally explain the H2 result above – the exact per-trial trigger condition is a complex function of window-slot reuse and devblock-array-shift interleaving that this investigation did not instrument further, since it is not required to confirm or fix the defect. The defect is independent of, and predates, unrelated same-day work on Artemis's disk-format detection (`BLK-CONFIRM-FORMAT`) – none of that work touches `cache_get_slot` or the window-cache functions.

5 Fix

Three call sites in `block_words.c` wrote through the stored window pointer before calling `blk_update()`: `block_word_update` (backing the Forth word `UPDATE`), `blk_vm_evict`'s all-slots-dirty fallback, and

`blk_vm_flush_all` (backing FLUSH/SAVE-BUFFERS). Each now re-resolves a *guaranteed-current* pointer by logical block number immediately before use:

```
uint8_t *fresh = blk_get_buffer((uint32_t) blk, 1);
if (!fresh) { vm->error = 1; return; }
memcpy(fresh, vm->memory + base, BLOCK_SIZE);
vm->blk_vm_cbuf[s] = fresh; /* keep the window's own copy in sync */
```

`blk_get_buffer()` internally calls `cache_load_devblock()`, which correctly resolves (or reloads) whichever slot currently holds that logical block number, however the array has shifted since the window pointer was first captured. The fix is confined entirely to the consumer side and does not alter the block subsystem’s cache architecture: never trust a raw cache pointer across an operation boundary where the producer may have invalidated it – ask for the current one again, right before using it.

6 Verification

The fix was verified by re-running the identical stress test (same seed, same 50-trial coprime-stride sequence) on all three supported architectures.

Run	Trials	Passed	Failed
amd64 (pre-fix)	50	25	25
amd64 (post-fix)	50	50	0
aarch64 (post-fix)	50	50	0
riscv64 (post-fix)	50	50	0

Table 2: Surface stress test results before and after the fix.

Pooling the three post-fix runs against the pre-fix run and testing H_0 : “pre-fix and post-fix trials share the same true pass rate” with a two-proportion test rejects H_0 at $p = 2.04 \times 10^{-19}$ (Table 1, H3; 95% CI on the difference in pass rate: $[-0.652, -0.348]$). This is the definitive result in this report: unlike H1/H1b/H2/H2b, which narrow the search space without identifying a mechanism, H3 directly confirms that the change made in Section 5 – and only that change – accounts for the difference between a 50% and a 100% pass rate, consistently, across three independent instruction-set architectures.

All three post-fix architectures also completed the standard acceptance boot to `ok>` with the fix in place, and Hera’s dictionary-hash parity across amd64 and aarch64 matched exactly for identical capsule content, confirming the fix does not disturb cross-architecture determinism.

7 Disposition

The surface stress test is now a permanent fixture of the Artemis capsule (`capsules/artemis/init.4th`, blocks 4160–4173), invocable on demand as `ART-STRESS` (fifty trials, fixed seed) or `<seed> <reps> ART-STRESS-TEST` for a custom run. It is not wired into the automatic boot sequence – block 4170 carries the entry point commented out (`\ ART-STRESS`), matching the same disable-by-default convention already used by `ACL.4th`’s self-activation toggle, so ordinary boots pay no extra time for it. A future maintainer can uncomment that one line to run it automatically, or invoke it manually from the console at any time. The unresolved H2 finding (Section 4) is left as an open question for a future investigation with additional cache-state instrumentation, not as a loose end in the fix itself, which H3 confirms is complete.