PURPOSE-BUILT ANDROID / ROOT-CAUSE INVESTIGATION

THE OOM THAT WASN'T

A production Android 16 device kept restarting its core system process under load, blocking a clean compatibility pass. Every signal pointed to an out-of-memory leak. It wasn't one, and the real cause was a single defect with a known upstream fix.

CLIENT	SECTOR	ENGAGEMENT	OUTCOME
Confidential	Purpose-built Android	CTS engineering investigation	Root cause determined

THE CHALLENGE

A core process kept dying under load

During compatibility testing, the device's central Android framework process grew towards 69 GB of virtual size and terminated, restarting system services mid-test and leaving runs unable to complete. The failure was load-dependent and reset itself on reboot, the hardest kind to pin down. The build was a locked production image with no root, so the usual live memory diagnostics were off the table. On its face it read as an out-of-memory leak, with the Java heap and a graphics-buffer leak both waiting to take the blame.

THE APPROACH

Forensics, not guesswork

B8N worked three evidence paths together: a privileged postmortem tombstone with the full process memory map, live read-only device diagnostics, and official AOSP source research. Rather than accept the size, we reconciled it. A captured memory map of 66,120 mappings, most of them leaked Icing search-index files, resolved to a single repeating unit: roughly 42 mappings and 20 file descriptors leaked on every failed attempt, across about 1,414 attempts. The Java-heap and graphics-buffer hypotheses were ruled out on the evidence, and the captured failure stack was matched to specific AppSearch source revisions.

THE OUTCOME

One defect, and the exact fix

~69 GB	1	2
PHANTOM VIRTUAL MEMORY	ROOT-CAUSE DEFECT	UPSTREAM FIXES NAMED
File-backed, not a real OOM	Evidence traced to source	Down to the feature flag

The cause was not memory exhaustion but an AppSearch failure-path resource leak: a corrupt per-user store made search-instance creation fail, and the failing path left its native search engine mapped and open on every retry until the process hit its mapping limit and aborted. Ordinary app installs retrigger it, so it was a product defect, not a test artefact. B8N placed the shipped module in the regression window between a caching change and Google's two corrective commits, and named the fix: both commits plus the AppSearch close-on-failure flag, present from Android 16 QPR1 but not the shipped branch. A mysterious, intermittent crash became a one-line root cause with a verifiable fix and acceptance criteria, before it reached the field.

METHOD AND REFERENCES

Determined from a privileged postmortem, live read-only diagnostics and official AOSP source history. Client, device and build details withheld under confidentiality.