

# Automating Linux Kernel Crash Analysis with LLMs

---

Chris Arges [carges@cloudflare.com](mailto:carges@cloudflare.com)

# Introduction

- **Who** Chris Arges, Senior Systems Engineer, Cloudflare Linux Team
- **What** LLMs to automate Linux kernel workflows at scale
  - background, the human workflow, the problem
  - what we tried, what we learned, what we're building

# Cloudflare and our network

- Linux across **330+ cities**, many servers per datacenter
- Build our own latest **longterm stable** kernels
- Continually update to the latest patch, update to next LTS every year
- Releases are done gradually and slowly, many versions running at once
- At this scale, **kernel crashes are inevitable**



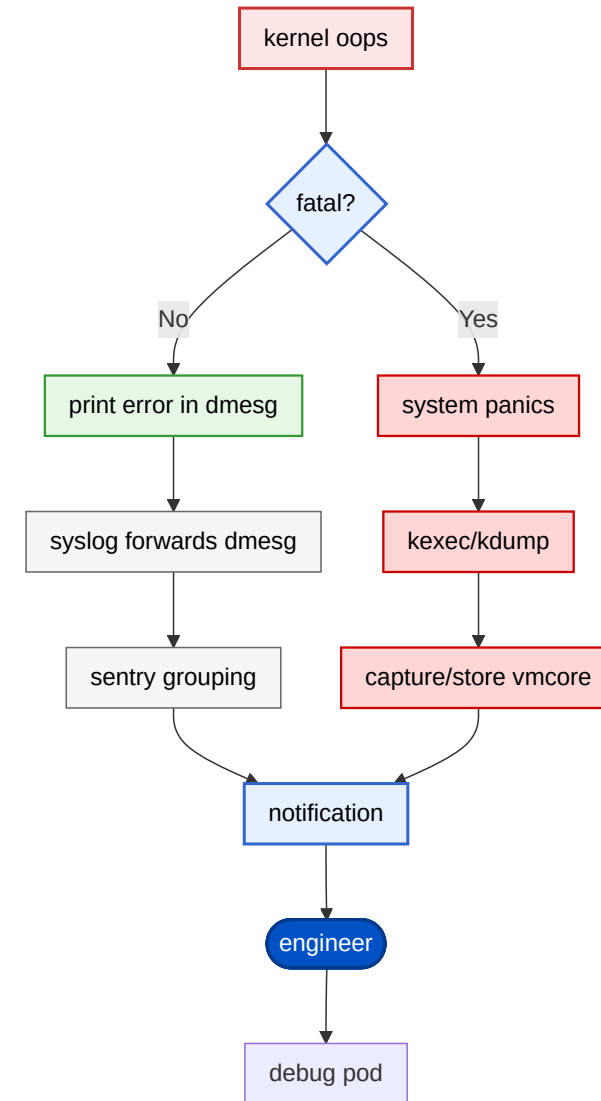
Source: [cloudflare.com/network](https://cloudflare.com/network)

# Types of Kernel Ooops/Errors

- **Kernel panics:** fatal, system halts; kdump/kexec captures a `vmcore`
- **Ooopses:** recoverable (NULL deref, invalid opcode, `BUG_ON()`); stack trace in `dmesg`
- **Warnings:** `WARN_ON()`, gives a stacktrace but continues execution
- **KASAN warnings:** sanitizer run on some machines to catch memory bugs early
- **Soft lockups & hung tasks:** CPU stuck or task blocked >120s; watchdog emits traces
- **MCEs:** hardware CPU/memory errors; usually hardware not software issues

# How Ooopses Flow to Engineers

- **dmesg** → **syslog** → **Sentry**: aggregates, deduplicates, groups events fleet-wide
- **Panic** → **kdump/kexec**: second kernel boots, dumps full memory image as **vmcore**
  - encrypted core sent to bucket
- Engineers access vmcores through **debug pods**
  - contains vmcore, crash, debug symbols
- **Sentry notifications** arrive in the team channel



# Anatomy of a crashdump

- The **crash** utility: interactive debugger for vmcores, wraps gdb
  - inspect memory, walk data structures, decode backtraces

```
KERNEL: /usr/lib/debug/vmlinux-6.18.21-cloudflare-kasan-2026.4.6
DUMPFILE: vmcore-131m47-6.18.21-cloudflare-kasan-2026.4.6-9c863760-df0d-45f4-890e-80dd297db270 [PARTIAL DUMP]
CPUS: 128
DATE: Thu Jan  1 00:00:00 UTC 1970
UPTIME: 2 days, 17:31:12
LOAD AVERAGE: 1377.14, 1265.63, 568.87
TASKS: 24629
NODENAME: testmachine1
RELEASE: 6.18.21-cloudflare-kasan-2026.4.6
VERSION: #1 SMP PREEMPT_DYNAMIC Mon Sep 27 00:00:00 UTC 2010
MACHINE: x86_64 (2000 Mhz)
MEMORY: 383.9 GB
PANIC: "Oops: Oops: 0002 [#1] SMP KASAN NOPTI" (check log for details)
PID: 1727261
COMMAND: "kworker/u530:3"
TASK: ffff88c8d0c8c680 [THREAD_INFO: ffff88c8d0c8c680]
CPU: 66
STACK DUMPING (ENABLED)
```

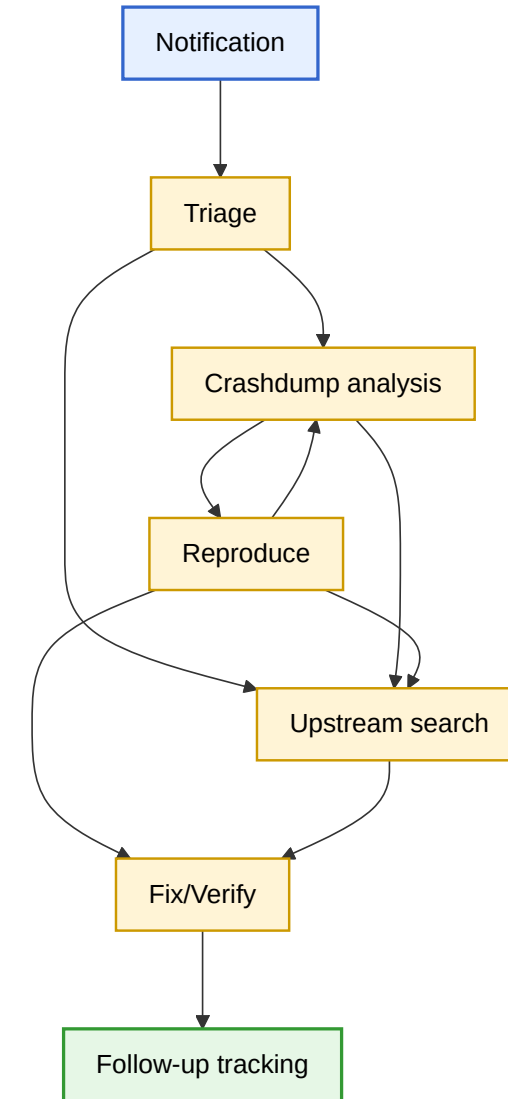
# How we do things today

---



# How Engineers Handle a Notification

1. **Notification:** alert arrives. Is it new? Serious?
2. **Triage:** dedupe against known issues; decide whether to investigate
3. **Crashdump analysis:** debug pod, load vmcore, run crash, read stack trace, poke memory, debug and iterate
4. **Upstream search:** `git log --grep`, LKML archives, mailing lists; already fixed?
5. **Reproduce:** write a C reproducer, boot test kernels (vng/QEMU), iterate
6. **Fix:** write patch, upstream review, backport, deploy, verify
7. **Follow-up tracking:** landed? deployed? crash rate dropped? regressions?



# The challenges we have

**1 million** sentry

events/month

**80** merged issues/month

**60** bugs in the JIRA backlog

very few people on Linux team

*...and we have feature work too*

## Where the time goes

- **Triage is hard:** Sentry merges by backtrace, but those vary across kernels; similar traces can have different root causes (and vice-versa)
- **Upstream search takes time:** many mailing lists, multiple trees (mainline / stable / subsystem); manual searches miss things
- **Repro is non-deterministic:** often needs specific hardware, memory pressure, network load, BPF programs
- **Follow-ups slip:** did the backport land? deploy go out? crash rate actually drop?
- **Knowledge doesn't scale:** crash debugging has a steep learning curve, so does working with upstream

# Let's throw AI at this problem

---

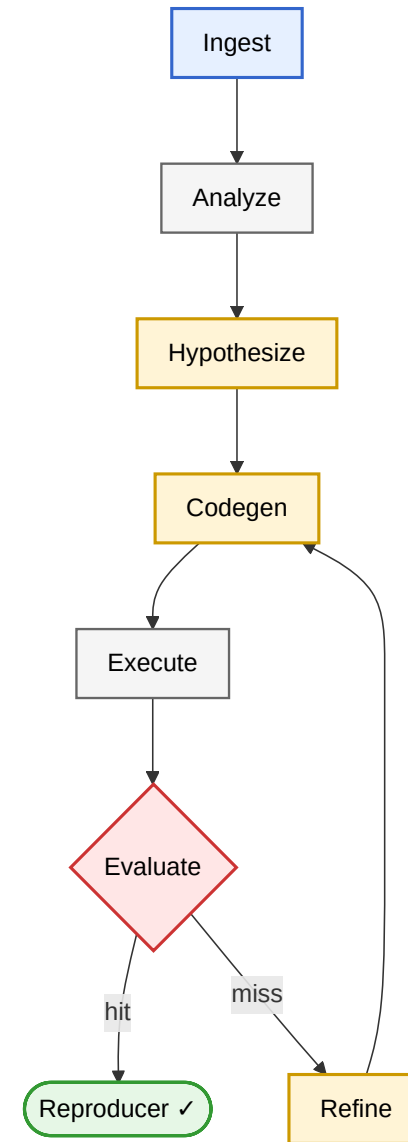


# Approach 1: Naive interactive agent

- Started by using Opendcode CLI and have it "learn by doing"
- Teach the LLM to **drive** `crash` interactively (over SSH via tmux)
- Added upstream Linux skill files as well
- Encoded learnings as **skills**: crashdump analysis, kernel fundamentals
  - As simple as "create a skill based on what you've learned so far"
- **Successes**
  - Found a real bug, and proposed a patch.
  - Patch was nearly identical to one just posted upstream
- **Learnings**
  - We need to search lore mirrors before proposing a patch
  - Reproducing and testing were missing, need better evidence

# Approach 2: Python harness for a reproducer

- **Python-driven pipeline**, LLM as a **constrained subprocess** (not the orchestrator)
- BPF tracer is used to see if the reproducer hit the *right* code path, not just *a* crash
- **Successes**
  - Systematic, reproducible runs with real output
  - Evidence-driven refinement loop
- **Learnings**
  - Time consuming, sometimes gets stuck
  - Needs more human in the loop to help guide process



# Supporting Tools

- **Shared Reproducers**
  - Central repo for reproducers across the team
  - Shared reproducibility artifacts → easy to re-use approaches and scaffolding
- **Shared Skills**
  - Team repo for skills; everyone benefits from each other's work
  - crashdump · server-crashes · reproducer · virtme-ng
- **Mailing List Mirror**
  - local lore mirror so agents can easily search

# Case Studies

---

## Case 1: filemap\_fault race

- Agent analyzed vmcore → identified a **race between folio lookup and folio split**
- Produced root-cause analysis with **evidence**:
  - register values, struct contents, memory state
- Reproducer was difficult to make
  - needed kernel modifications to widen the race window
- Proposed a **patch** after testing internally with some success
- Feedback from this RFC was helpful, ultimately another fix addressed this issue

## Case 2: KASAN Out-Of-Bounds in `symbol_string`

- Agent identified **two distinct bugs** where humans initially saw one
- Automated tedious parts: pointer-chasing, stack-frame diagrams
- Used automation on volume of reports to help find bad values in `modname` related to BPF
- Built a working reproducer after the breakthrough
- Ultimately we found some **patches** that needed backporting.

# What did we learn?

---

## Coordinate: design before building

- **Everyone tried different things:** great for exploration, but we re-hashed the same problems
- **Think about your workflow first:** then build the tool
- Don't just wrap an LLM around a problem and hope

## Skills: living documents

- **Skills need ongoing improvement** — they decay as kernels and tools change
- You don't have to write them from scratch:
  - **Ask the LLM to write down what you just learned**
- Compounding returns: every investigation can feed back into the toolkit
- Need to periodically prune and maintain: more text == using more context window

# LLMs can be confidently wrong

"This is the **root cause**."

"This is the **smoking gun**."

"That **confirms** it."

"Now I have the **complete** picture."

- Be skeptical! Demand **evidence**, not confidence
- Insist on proof through **testing**
- Treat every claim as a hypothesis until verified

# Input data matters

- Use **multiple sources**:
  - crash report
  - server metrics
  - crashdump analysis
  - mailing lists
  - Sentry patterns
- **Record bad hypotheses** so the agent doesn't keep retrying them
- Prune the search space deliberately

## Use different models

- A model may get 'stuck', or hypothesis generation may need more variability
- It's like having different people look at the bug
- Different training data, different weaknesses, different strengths

## Safe agentic execution is critical

- Let the agent **safely execute programs** on your behalf
- But use **protection**: sandboxing, VMs
- Being able to **compile a kernel** is critical
- `vng` (**virtme-ng**) is amazing for fast kernel boot/test

## Script repeatable tasks

- If a task is repeatable, **script it**
- If the agent can write the script, have it do so
- Keep **honing the script**
- Saves tokens, improves determinism, becomes part of the toolkit

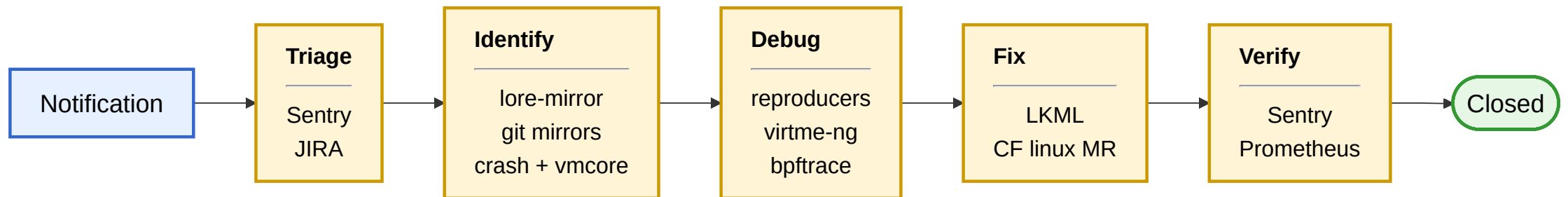
# Let's make this work better

---



# The unified workflow "kdoc"

Combined end-to-end workflow · `cd` into the repo, agent loads skills · humans guide progression



# Summary of the work

- **Skills** encode expert knowledge: crashdump · server-crashes · reproducer · virtme-ng · upstream search
- **Commands** provide structured entry points for common tasks
- **Reproducer tools** Use python harness, plus info from crashdump analysis
- **Interactive agent** for exploratory analysis with human in the loop
- **Shared repos** (skills + reproducers) so the team compounds learnings
- All this lives in a single project "kdoc"

# kdoc

kdoc -- Kernel Crash Triage Toolkit

## SETUP

/kdoc-setup	Configure repos, install skills, validate environment
/kdoc-setup check	Read-only health check
/kdoc-help	This reference

## TRIAGE (Phase 1)

/kdoc-triage N	Scan Sentry for top N unticketed crashes (default: 5)
----------------	---

## IDENTIFY (Phase 2)

/kdoc-investigate ID	Full investigation of a Sentry issue (e.g., KERNEL-12345)
/kdoc-search-upstream T	Search ML archives + upstream tree for fixes
/kdoc-backport-check C	Check if an upstream commit is in our kernel
/kdoc-update-jira KRN-X	Push analysis findings to a JIRA ticket

## DEBUG (Phase 3)

/kdoc-analyze-dump SSH	Analyze a vmcore on a crash dump debug pod
/kdoc-reproduce KRN-X	Build reproducer, boot test VM, verify fix

## FIX & FOLLOW-UP (Phases 4-5, stubs)

/kdoc-submit-fix	Submit patch MR to CF kernel repo
/kdoc-draft-upstream	Draft upstream bug report or patch email
/kdoc-verify	Confirm a deployed fix resolved the crash

## TYPICAL FLOW

```
/kdoc-triage → /kdoc-investigate → /kdoc-search-upstream  
→ /kdoc-analyze-dump → /kdoc-reproduce → /kdoc-update-jira
```

FIRST TIME? Run /kdoc-setup.

# kdoc: Reauthenticating MCP

- Multiple Sentry issues may be the same bug. Use `bootuuid` and `uname.machine` to identify same-incident issues.
- The KASAN event title may not reflect the actual fatal crash. Note this if the kernel release string contains "kasan".
- Always end with concrete next-step commands the user can copy-paste.

```
⊙ cf-portal_sentry-mcp_get_sentry_issue [issueId=38342738]
Unauthorized

⊙ cf-portal_jira-mcp-server_get_jira_ticket_info
Unauthorized

⊙ cf-portal_sentry-mcp_list_sentry_issues [query=KERNEL-3Q5R, limit=1]
Unauthorized

⊙ cf-portal_jira-mcp-server_get_jira_ticket_info
Unauthorized
```

Both MCPs returning Unauthorized – auth token may have expired. Let me retry once.

```
⊙ cf-portal_sentry-mcp_list_sentry_projects [limit=1]
Unauthorized
```

The Sentry and JIRA MCP sessions both returned Unauthorized – the auth tokens have expired (or the upstream MCPs went idle). I can't proceed without them.

**Action needed:** re-authenticate the MCPs and re-run.

You can either:

1. Use the portal to re-auth in one shot:  
`/portal_toggle_servers`  
then re-enable `sentry` and `jira-mcp-server` (this should trigger fresh OAuth).
2. Or toggle each one individually:  
`/portal_toggle_single_server sentry-mcp untoggle`  
`/portal_toggle_single_server sentry-mcp toggle`  
`/portal_toggle_single_server jira-mcp-server untoggle`  
`/portal_toggle_single_server jira-mcp-server toggle`

Once both come back authorized, re-run:  
`/kdoc-investigate KERNEL-3Q5R`

and I'll do the full investigation (fetch the issue + latest event, pull KRN-1260 details, compare stack traces, search related Sentry issues, and decide whether KERNEL-3Q5R belongs in KRN-1260 or needs its own ticket).

■ Build · Claude Opus 4.7 · 29.6s



Build · Claude Opus 4.7 Anthropic: Cloudflare AI Gateway

## Kernel crash triage from Sentry

### Context

209,078 tokens  
21% used

### MCP

- cf-portal Connected
- chrome-devtools Connected

### LSP

LSPs are disabled

29

~/cf-repos/kdoc:main

• OpenCode 1.14.48

209.1K

ctrl+p commands

# kdoc: Triage

```
ctrl+x down view subagents
```

Excellent intel. Key findings consolidated:

- **KERNEL-3Q5G** is a NEW `skb_data_move` WARN signature: `bpf_skb_change_head` from LWT BPF (`send_to_` ). KRN-1245 is the existing parent issue (KERNEL-3PYD). Need to compare top frames – likely **dup of KRN-1245**.
- **KERNEL-3PYF (KRN-1244, Bad page state)** + **KERNEL-3Q1G (soft lockup with [B]=BAD\_PAGE)** are likely a **fleet-wide BAD\_PAGE cluster** on `6.18.24-cloudflare-2026.4.37` and `6.18.25-cloudflare-kasan-2026.4.41`. KERNEL-3Q1G should be grouped under KRN-1244.

Final candidate selection (top 5 non-MCE crashes from 6.18, prioritizing ranks):

#	Tier	Sentry ID	Title	Count	KRN Ticket	Action
1	CRIT	KERNEL-3PYF	BUG: Bad page state in process logger	100	KRN-1244	Already ticketed
2	MED	KERNEL-3PQD	KASAN stack-00B unwind_next_frame	479	KRN-1255, KRN-1224	Already ticketed
3	MED	KERNEL-3PYD	WARN skb_data_move	1266	KRN-1245	Already ticketed
4	MED	KERNEL-3Q5G	WARN skb_data_move CPU#76	35	NO	Group with KRN-1245 (dup)
5	MED	KERNEL-3Q1G	soft lockup CPU#88 prometheus-ebpf	2	NO	Group with KRN-1244 (BAD_PAGE cluster)
6	MED	KERNEL-3Q1S	WARN scx_cgroup_move_task	2	KRN-1260	Already ticketed
7	MED	KERNEL-3NNC	WARN get_rps_cpu	55918	KRN-1069	Already ticketed

Unticketed crash-level candidates that fit the user's request for "top recent": **KERNEL-3Q5G** and **KERNEL-3Q1G**. Only 2 unticketed crashes exist in the 6.18 cohort. The rest of the high-event-count issues already have JIRA tickets.

Let me write the triage log now.

```
# Prepare triage log directory
$ mkdir -p triage-log && date +%Y-%m-%d
2026-05-12
```

```
# Check existing triage logs
$ ls triage-log/ 2>/dev/null
```

```
□
```

```
Build · Claude Opus 4.7 Anthropic: Cloudflare AI Gateway
```

```
..... esc interrupt
```

```
184.3K (18%) ·
```

```
ctrl+p commands
```

## Kernel crash triage from Sentry

### Context

184,280 tokens  
18% used

### MCP

- cf-portal Connected
- chrome-devtools Connected

### LSP

LSPs are disabled

30

```
~/cf-repos/kdoc:main
```

```
• OpenCode 1.14.48
```

# kdoc: Triage Findings

```
199 ## Notes
200
201 - The 6.18 fleet appears to have **two active crash clusters**: (a) the `bad_page` cluster (KRN-1244 + KRN-1243 + KERNEL-3Q1G as a
downstream symptom), and (b) the LWT-BPF `skb_data_move` WARN cluster (KRN-1245 + KERNEL-3Q5G).
202 - KRN-1255 (KASAN unwind_next_frame) is the highest-volume issue but is on the `cloudflare-kasan` debug build, not the production
kernel.
203 - One MCE was filtered out: KERNEL-3Q1F on GIGABYTE hardware (same hardware platform that has historically had cstate-related
machine checks).
204 - A large cohort of ~35 boot-banner / driver-probe info events (KERNEL-3Q3*-KERNEL-3Q5*) clustered at 2026-05-12 09:33-11:00 UTC
corresponds to the rollout of `6.18.25-cloudflare-2026.4.41` and `6.18.29-cloudflare-2026.5.5` - these are normal boot noise, not
crashes.
205
206 ---
207
208 Generated by [kdoc](https://gitlab.cfddata.org/cloudflare-developers-scratchpad/krn/kdoc).
209
```

Triage saved to `triage-log/2026-05-12-triage.md`. Summary:

**6.18 series, top 5 non-MCE crashes triaged.** Only 2 unticketed crash-level candidates exist in the current 100-issue window – the rest of the high-event-count issues are already in JIRA.

## Unticketed candidates needing action:

- [WARN\\_skbuff.h:4609 / skb\\_data\\_move \(LWT BPF\) KERNEL-3Q5G](#) (35 events, escalating today)
  - Same file:line + same `bpf_skb_change_head` LWT-BPF caller as **KRN-1245** (parent issue KERNEL-3PYD, 1266 events). Sentry split them by CPU#/PID variance.
  - **Action:** Merge into KERNEL-3PYD, update KRN-1245.
- [soft lockup / smp\\_call\\_function \(BAD\\_PAGE-tainted\) KERNEL-3Q1G](#) (2 events, fatal, same-incident cluster)
  - `[B]=BAD_PAGE` taint already set when the lockup hit on `l3les1` running the KASAN 6.18.25 build. Stack: `madvise → flush_tlb_mm_range → smp_call_function_many_cond` stuck.
  - Likely downstream of **KRN-1244** (BUG: Bad page state). Same kernel build family.
  - **Action:** Group with KRN-1244 or open new ticket if dump confirms it's an independent TLB-shutdown stall.

**MCE filtered out:** KERNEL-3Q1F (Hard LOCKUP on GIGABYTE F hardware) `isMce:true`, hardware fault – route to HW team).

**Already-ticketed (no action):** KRN-1244, KRN-1255/1224, KRN-1245, KRN-1069, KRN-1260, KRN-1259, KRN-1206.

Next steps you may want:

- `/kdoc-update-jira KRN-1245` – note the KERNEL-3Q5G merge candidate (+35 events)
- `/kdoc-update-jira KRN-1244` – link KERNEL-3Q1G as a same-class secondary symptom
- `/kdoc-investigate KERNEL-3Q1G` – pull full kernel log to confirm the BAD\_PAGE precedence

■ Build · Claude Opus 4.7 · 14m 53s



Build · Claude Opus 4.7 Anthropic: Cloudflare AI Gateway

## Kernel crash triage from Sentry

### Context

192,045 tokens  
19% used

### MCP

- cf-portal Connected
- chrome-devtools Connected

### LSP

LSPs are disabled

31

~/cf-repos/kdoc:main

• **OpenCode** 1.14.48

192.0K (19%) ·

ctrl+p commands

# kdoc: Investigate

## Crash Classification

- **Type:** BPF program runtime error – `scx_bpf_error(-ENOMEM)` from `lavd_cgroup_init`, triggers full scheduler disable
- **Subsystem:** `sched_ext` (`scx_lavd` BPF scheduler – userspace project from `sched-ext/scx`)
- **Severity:** **HIGH-impact, MEDIUM-priority** – every affected host loses its BPF scheduler and falls back to CFS. Not a crash, not a kernel correctness issue, but degrades the `scx_lavd` experiment outcome.
- **MCE:** no

## Current Status

- **JIRA ticket:** needs creation (not the same bug as KRN-1260)
- **Upstream kernel fix:** not applicable – kernel side is correct, fix belongs in the `lavd` BPF program
- **Fleet exposure:** 6+ hosts confirmed across both arches and both `6.18.24` and `6.18.25` production kernels (plus one KASAN-debug host) – likely much wider; this only became visible 2026-05-12 11:24 UTC, which roughly matches the rollout window of `6.18.25-cloudflare-2026.4.41`

## What likely changed

The fleet-wide first-seen timestamp of 11:24 UTC on 2026-05-12 coincides with the `6.18.25-cloudflare-2026.4.41` rollout (per the boot-banner cohort in the morning's triage). The `lavd` version `1.0.20_g85f140fd_dirty` is identical across all events, so **lavd itself is not new** – but the bug only surfaces on whatever specific cgroup pattern these hosts are creating now. Likely candidates:

- A workload started creating short-lived cgroups (e.g., systemd transient scopes, container churn) that hammer `lavd_cgroup_init` faster than the BPF prog's allocator can keep up.
- A `bpf_obj_new/bpf_kfunc` allocation pool inside the `lavd` BPF program is exhausted under high cgroup-creation rate.
- KASAN build (`131dm7`) hitting it confirms it's not a memory-overcommit issue on the host – it's specifically the BPF prog returning `ENOMEM` internally.

## Recommended Actions

1. **Create a new KRN ticket** for `scx_lavd_cgroup_init -ENOMEM disabling scheduler fleet-wide on 6.18.24/25`. Mention KRN-1260 as related-not-duplicate, link KRN-1192 as the deployment owner. Owner candidate: `_` (owns KRN-1192 / `scx_lavd` experiment).
2. **Merge KERNEL-3Q5R and KERNEL-3Q5B in Sentry** – same bug, only the title arch token differs. They'll keep splitting on every new arch otherwise.
  - URL to view both at once: <https://sentry10.cfdata.org/organizations/cloudflare/issues/?project=248&query=issue%3A%5BKERNEL-3Q5R%2C+KERNEL-3Q5B%5D>
  - Or by message: [https://sentry10.cfdata.org/organizations/cloudflare/issues/?project=248&query=lavd\\_cgroup\\_init](https://sentry10.cfdata.org/organizations/cloudflare/issues/?project=248&query=lavd_cgroup_init)
3. **Inspect lavd source at `main.bpf.c:1978`** – file the actual bug there, not in the kernel. The [sched-ext/scx repo](https://github.com/sched-ext/scx) (<https://github.com/sched-ext/scx>) is the upstream; check whether `1.0.20` is current and whether there's a newer release with this `ENOMEM` path fixed. If our pin is `g85f140fd_dirty`, we likely have local modifications too.
4. **Capture an event from `131dm7`** (the KASAN-debug host running `6.18.25-cloudflare-kasan-2026.4.41`) to confirm no KASAN report precedes the disable – KASAN would catch any BPF-allocator misuse if one exists.

## Copy-pasteable commands

```
# Update the new KRN ticket (after creating it via Sentry's "Link Issue" UI on KERNEL-3Q5R)
/kdoc-update-jira KRN-XXXX
```



Build · Claude Opus 4.7 Anthropic: Cloudflare AI Gateway

## Kernel crash triage from Sentry

### Context

277,373 tokens  
28% used

### MCP

- cf-portal Connected
- chrome-devtools Connected

### LSP

LSPs are disabled

32

~/cf-repos/kdoc:main

• **OpenCode** 1.14.48

277.4K (28%) · `ctrl+p` commands

# kdoc: Crashdump Analysis

Given `CONFIG_NET_DEVMEM=y` and that `...` is a veth attached to a process that's likely doing devmem-TCP or similar `IO_URING` zero-copy `net_iov` work – the `net_iov` hypothesis is strong.

Let me check the frag pointer on the stack (at `0x8(%rsp)` of the `__xsk_rcv` frame):

```
# Read stack values at __xsk_rcv frame
$ /tmp/crash-cmd.sh "rd ffffa49cecl7a28 8" 3

ffffa49cecl7da0: 000082720de6edc8 ffffa49cecl7e00
ffffa49cecl7db0: ffffa49cecl7e30 ffff909158c8c400
ffffa49cecl7dc0: ffffffff090f479 0000000000000000
ffffa49cecl7dd0: 0000000000000000 ffff90b7c8668540
ffffa49cecl7de0: 0000000000000002 ffff90b7c8668540
ffffa49cecl7df0: ffffa49cecl7e2f ffff90af90178000
ffffa49cecl7e00: 00000000ffffffff ffffffff900b9775
#14 [ffffa49cecl7e08] __napi_poll at ffffffff900b9775
ffffa49cecl7e10: ffffffff900b9aa4
#15 [ffffa49cecl7e10] napi_threaded_poll_loop at ffffffff900b9aa4
...
```

Click to expand

Good – at `ffffa49cecl7a38` (which is `0x18(%rsp)` of `__xsk_rcv`): `ffff9093083acc00` is the `xdp_buff` pointer (saved `%rcx`). The next stack slot `0x20(%rsp)`: `ffffa49d48c0a100` is the destination buffer. At `0x8(%rsp)` for the frag: would be `ffffa49d48c0a100` actually no – let me look. Address `ffffa49cecl7a28` is `0x8(%rsp)`. Value = `ffffffff000005f6`. That looks weird (not a kernel pointer). Hmm – that's actually a value, not a `skb_frag_t*`. Probably gone stale or different stack slot.

Actually – wait. I need to re-examine which slot is what. Stack frame after `sub $0x30,%rsp` from RIP starts at the `callee` frame. The current RSP (at `memcpy` entry) is `ffffa49cecl7a20`. The function reserves `0x30` bytes for locals, plus 6 callee-saved register pushes (48 bytes). The frame layout: when `memcpy` was called, RSP =



Build · Claude Opus 4.7 Anthropic: Cloudflare AI Gateway

## Vmcore crash dump analysis

### Context

294,208 tokens  
29% used

### MCP

- cf-portal Connected
- chrome-devtools Connected

### LSP

LSPs are disabled

# kdoc: Crashdump Details

- Same task ( 's veth)
- Same kernel class (6.12.x cloudflare)
- Same root cause already established in the ticket: `skb_frag_address()` returns NULL because the frag's `netmem` has the low bit set (`net_iov`), and `xsk_copy_xdp` blindly `memcpy`s from NULL. confirmed via crash that the frag had `netmem = 0xfffffa3e00000001` (bit 0 = `net_iov`), and the underlying `net_iov` had `pp_ref_count = 0` – i.e., a page that was returned to the allocator and is being referenced via a stale frag.

My analysis matched, but the ticket already has it nailed down further:

1. Triggered by `flowtrackd`, which is the only product redirecting on a veth via `AF_XDP`. UDP fragments were the trigger.
2. Partial mitigation deployed: | 2025.10.1 (MR 629) stopped sending UDP fragments to userspace as of 2025-11-03.
3. But crashes are still happening – reported recurrence in dub02 on 2025-11-08, ARM nodes also crashing (| , 2025-11-10), and now your 766m80 crash on 2026-05-12 in the dump I just analyzed.

The bug is still open (In Progress, P4) and ours is a fresh occurrence to add. The new evidence is meaningful:

- This crash is on `6.12.85-cloudflare-2026.4.44` – ~6 months and many kernel rebuilds after the original `6.12.47` reports, so the bug persists in current production kernels.
- `766m80` is a new host signature ( , vs the earlier reports). Different DC than dub01/dub02.
- Confirms the `flowtrackd` workaround didn't fully eliminate the bug (or there's another product path now hitting it).

I should update my analysis file to cross-reference KRN-1138 instead of treating this as a novel finding, and ideally post a comment on KRN-1138 with this new occurrence. Want me to:

1. Update `analyses/766m80-2026-05-12.md` to link to KRN-1138 and reframe as "new occurrence of known bug"
2. Run `/kdoc-update-jira KRN-1138` to post a comment on the ticket with the 766m80 vmcore artifact ID and the fact that the bug is still hitting 6.12.85 (post-`flowtrackd` mitigation)

Or both?

■ Build · Claude Opus 4.7 · 26.4s

```
/kdoc-update-jira KRN-1138
```

Build · Claude Opus 4.7 Anthropic: Cloudflare AI Gateway

294.2K (29%) ·

ctrl+p commands

## Vmcore crash dump analysis

### Context

294,208 tokens  
29% used

### MCP

- cf-portal Connected
- chrome-devtools Connected

### LSP

LSPs are disabled

34

~/cf-repos/kdoc:main

• OpenCode 1.14.48

# kdoc: JIRA Posted

```
1978}}, not in the kernel. The prog's {{cgroup_init}} op returns {{-ENOMEM}} on a userspace {{mkdirat(cgroupfs)}}, then calls {{scx_bpf_error_bstr}}, which disables the whole scheduler and falls back to CFS.
```

```
* Fleet-wide on the {{6.18.25-cloudflare-2026.4.41}} rollout (first event 2026-05-12 11:24Z, ~7h after rollout). Also seen on {{6.18.24-cloudflare-2026.4.37}} and the kasan variant on 131dm7. Hosts so far: 693dm5, 774dm1, 504dm8, 693dm3, 539dm10, 539dm11, 131dm7. Both arches. lavd build {{1.0.20_g85f140fd_dirty}} -- a {{dirty}} pin, so we may carry local changes.  
* x86_64 sibling Sentry issue [KERNEL-3Q5B|https://sentry10.cfdata.org/organizations/cloudflare/issues/38342144/] (20 events). Identical message, identical 21-frame kernel stack ({{scx_bpf_error_bstr -> bpf_prog*_lavd_cgroup_init -> scx_tg_online -> cgroup_mkdir}}), only the scheduler-name arch token differs. Merge into this ticket's Sentry issue.  
* Not the same bug as KRN-1260. KRN-1260 is a kernel-side {{WARN_ON_ONCE}} at {{kernel/sched/ext.c:3226}} on the {{sched_move_task}} path, non-fatal. This one fires from {{cgroup_mkdir}} and disables the scheduler.  
* Kernel side is correct. The two relevant upstream commits -- [568894edbe48 ("sched_ext: Add scx_cgroup_enabled to gate cgroup operations and fix scx_tg_online()")|https://github.com/torvalds/linux/commit/568894edbe48] and [ec010333ce7c ("sched_ext: scx_cgroup_exit() may be called without successful scx_cgroup_init()")|https://github.com/torvalds/linux/commit/ec010333ce7c] -- are both in v6.18. Fix belongs in [sched-ext/scx|https://github.com/sched-ext/scx].
```

```
[~mfleming] owns the scx_lavd deployment (KRN-1192); routing to you. Suggest checking what's at lavd {{main.bpf.c:1978}} on the {{g85f140fd}} pin and whether a newer scx release fixes this ENOMEM path.
```

```
----
```

```
_Generated by [kdoc|https://gitlab.cfdata.org/cloudflare-developers-scratchpad/krn/kdoc]._]
```

```
⊙ cf-portal_jira-mcp-server_get_jira_ticket_info
```

<b>Comment</b>	added (verified – visible on ticket, attribution footer present)
<b>Description</b>	unchanged (kept the Sentry stub intact; comment carries the analysis)
<b>Labels added</b>	none – MCP can't set labels. Suggested set for the UI: <code>bpf</code> , <code>sched-ext</code> (if exists; otherwise leave as-is)
<b>Links added</b>	2 auto-created Mention links – <b>KRN-1260</b> (routed as "different bug, same subsystem"), <b>KRN-1192</b> (lavd deployment owner). Sentry sibling <b>KERNEL-3Q5B</b> stays a markdown link in the comment, not a JIRA link.
<b>URL</b>	<a href="https://jira.cfdata.org/browse/KRN-1268">https://jira.cfdata.org/browse/KRN-1268</a> ( <a href="https://jira.cfdata.org/browse/KRN-1268">https://jira.cfdata.org/browse/KRN-1268</a> )

```
Follow-ups that need the UI (MCP can't do them):
```

- Merge Sentry **KERNEL-3Q5B** → **KERNEL-3Q5R** so the x86\_64 cohort folds into this ticket's primary Sentry issue.
- Optional labels: `bpf` from the kdoc skill set.

```
■ Build · Claude Opus 4.7 · 38.7s
```



```
Build · Claude Opus 4.7 Anthropic: Cloudflare AI Gateway
```

## Kernel crash triage from Sentry

### Context

```
277,373 tokens  
28% used
```

### MCP

- cf-portal Connected
- chrome-devtools Connected

### LSP

```
LSPs are disabled
```

```
~/cf-repos/kdoc:main
```

```
• OpenCode 1.14.48
```

# What's next

---

# Takeaways

1. LLMs are useful for kernel debugging **today**
2. Be deliberate: **design the workflow first**, then build tooling
3. **Encode expert knowledge as skills**: this is how you scale debugging across a team
4. Biggest wins are the **tedious parts**: vmcore analysis, pointer-chasing, dedup, upstream search
5. **Humans provide intuition; machines provide throughput**

## Next Steps

- **Semcode MCP:** Look at ways to leverage MCP for more effective code navigation
- **Patch writing and review:** Once we gain confidence, have system propose patches with human in the loop
- **Fully automated triage:** new crash arrives → agent dedupes, analyzes, files ticket with findings
- **Community opportunity:** skills and workflows are reusable beyond Cloudflare
- **Collaborate with others:** Work with other projects such as `linux-kernel-oops`

# Questions?

---