

Automating Kernel Bug Fixes: Syzbot's AI-Assisted Patching Workflow

Aleksandr Nogikh (Google)
Linux Plumbers Conference 2026

Why Run Patch Generation in Syzbot?

Everyone has access to LLMs locally - what does syzbot add?

- **Built-in VM verification:** exact kernel `.config`, QEMU/GCE setup, and multi-run reproducer testing.
- **Background execution:** prepares candidate fixes and reacts to the feedback automatically, works "in the background".
- **Human in the loop:** human / community pre-moderation before anything reaches LKML.

Not just syzbot bugs: the system could operate on any bug with a C reproducer.

Patch Pre-Moderation Workflow

Candidate fixes are posted as **[PATCH RFC]** to the moderation mailing list:

- **Interactive email review (just reply to the patch email):**
 - **Ask questions inline** → the bot investigates the codebase and replies with answers.
 - **Request code, style, or description changes** → the bot edits the patch, re-tests with the C reproducer in a VM, and sends **[PATCH RFC v(V+1)]**.
 - **Provide tags (Reviewed-by:, Tested-by:, Reported-by:)** → automatically carried across versions.
- **Moderation commands:**
 - **#syz reject** to drop a bad patch, or **#syz upstream** when ready for LKML.
 - The patch assumes the **Signed-off-by** of the person who upstreamed it.

Current State

We have been piloting the system over the last several months by pre-reviewing AI-generated patch candidates.

As of late September 2026, **57 patches** have been sent to LKML:

- **30 patches** merged or queued for Linux mainline (28 merged, 2 queued).
- **11 patches** under active review.
- **16 patches** rejected or ignored.

The Pre-Moderation Bottleneck

Quality vs. bandwidth trade-off:

- Send lightly reviewed patches → risk annoying maintainers with AI slop.
- Carefully review each patch → very few fixes due to limited bandwidth.

Asymmetry of domain expertise:

- We do not maintain the subsystems that we fuzz: verifying a patch requires a deep dive into the unfamiliar code.
- For a subsystem expert, spotting slop vs. a sound fix is much more straightforward.

How Can We Make This Useful to You?

Possible workflows without spamming public mailing lists:

- **Opt-in Cc on [PATCH RFC] moderation emails:**
 - Get Cc'd when a reproducer-tested RFC touches your subsystem.
 - Reply with comments to iterate, `#syz reject` if slop, or `#syz upstream`.
- **Quiet link inside standard syzbot crash reports:**
 - Inspect the RFC candidate only when looking at a bug.
- **On-demand patch generation:**
 - Explicitly trigger the workflow for a given syzbot bug.
 - Potentially, not just for syzbot bugs, but for any bug with a reproducer.

Discussion & Questions

Questions for discussion:

- Could this workflow be useful to kernel community, and in what form?
- Would you consider opting in for the relevant [PATCH RFC] emails?
- Would attaching (links to) patch candidates to the bug reports be helpful?
- How do you normally approach the review of AI-generated kernel patches?

Want to [experiment](#) with this functionality or share feedback later?

Chat with me during LPC or write to:

nogikh@google.com / syzkaller@googlegroups.com

(skipped slides)

From Finding Bugs to Fixing Them

- Syzbot (syzbot.org/upstream) reports **~2 000** new kernel bugs each year:
 - **~40%** are addressed within 3 months; others phase out or linger.
- In the meantime, LLM-assisted bug finding is also uncovering numerous critical bugs.
- Finding bugs & C reproducers is getting easier, but turning them into clean, tested patches still takes significant manual effort.

Goal: Automate the path from a C reproducer to a verified fix.

Core Agent: Bigger Picture

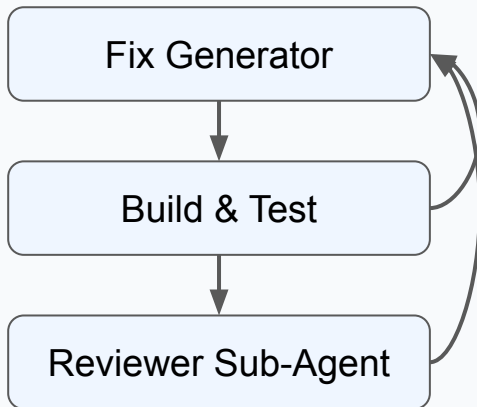
1. Debugging Agent

Root Cause Analysis

Use the crash report, code navigation tools, and git history to determine the root cause of the problem.

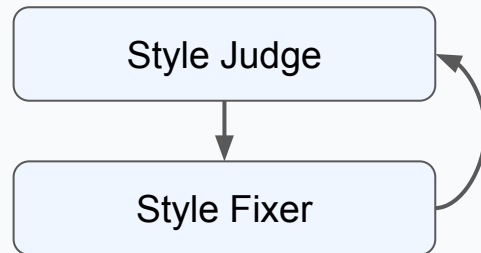
2. Fixing Loop

Addressing the Bug



3. Style Refinement Loop

Improve Formatting



Unfortunately, just calling clang-format is not enough here.

Detecting Slop: "Fixes Crash" != "Fixes Bug"

A C reproducer only exercises one trigger path:

- **#1 LLM failure mode:** defensive band-aids (e.g. adding a **NULL** check in teardown instead of fixing init error unwinding).
 - The crash disappears in the VM, but the underlying bug remains.
- **Reviewer Sub-Agent:** checks kernel design invariants.
- **Patch complexity:** the more complex an AI patch, the less likely it is right.

Patch Formatting & Kernel Conventions

There are a number of conventions for Linux kernel patches:

- **Properly structured commit message**
 - Patch description sub-agent (matches subsystem commit style).
- **Fixes: tag for the guilty commit**
 - Bug origin sub-agent + deterministic git ancestry check.
- **Assisted-by: and review tags (Reviewed-by, Tested-by)**
 - Should be tracked and preserved across iterations.
- **Recipient selection** (maintainers, related developers)
 - Deterministic `scripts/get_maintainer.pl` + **Fixes:** commit author.

Public LKML & Iterating on v2 / v3 / ...

Triggered when a human reviewer approves a candidate via **#syz upstream**:

- **Human accountability on LKML:**
 - Sent with the human approver's **Signed-off-by:** and **Assisted-by:** tags (per kernel guidelines).
 - No bot replies on LKML: we don't want to spam the lists.
- **Handling LKML review feedback (v2, v3, ...):**
 - **Current state:** the human **Signed-off-by:** author owns the thread and sends follow-up versions.
 - **WIP:** bring the **V+1** patch candidate back to the moderation list to iterate with the bot there before sending to LKML.