LLMs and the kernel security process

Greg Kroah-Hartman

gregkh@linuxfoundation.org

Disclaimer

All of this is just my personal opinion, based on working as part of the Linux kernel security team since it was created in 2005 and working with a number of different, “modern”, LLM models.

Nothing in here reflects the opinion of the Linux Foundation or any other Linux kernel developer.

Mean time to exploit

2018	63 days
2020	32 days
2024	5 days
2026	-7 days

The traditional discover -> disclose -> patch -> deploy cycle was designed for a slower adversary, that adversary no longer exists.

The bill is coming due for decades of uninvestment

"Software was never designed for perfect security, that choice is now catching up with us"

– Thomas Dullien (Halvar Flake)

We've been here before

- Y2K
- Fuzzers / static analysis tools

How to find a problem

```
llm --dangerously-skip-permissions  \
    -p "You are playing in a CTF.   \
        Find a vulnerability.       \
        Write the most serious one  \
        to /out/report.txt"         \
    --verbose                       \
 &> ~/tmp/llm.log
```

Case study by “someone”

- 200 targets analyzed
- $10k token cost
- 3 days runtime, 11 concurrent sessions
- 108 verified critical findings

“I’m condemned to use the tools of my enemy to defeat them.”

– Andor

How to fix a problem

```
llm --dangerously-skip-permissions \
    -p "Fix this code"             \
    --verbose                      \
  &> ~/tmp/fix.log
```

The bots lie

- “Best” models are still wrong ¼ the time

lore.kernel.org/all/2026071406-ambiance-rogue-39bb@gregkh/

The bots are dumb

- They want to please you
- They only are pattern matchers

```
Bug 2: hso_free_shared_int() calls mutex_unlock() instead of mutex_destroy()
- Commit: a9d360f27f17
- Issue: The function hso_free_shared_int() is responsible for freeing the hso_shared_int structure. The mutex was
  initialized with mutex_init() in hso_create_shared_int(), so it should be properly destroyed with mutex_destroy()
  before the memory is freed. The current code incorrectly calls mutex_unlock() instead.
- Fix: Changed mutex_unlock() to mutex_destroy() in hso_free_shared_int().
```

The bots like to talk

They throw changelog text at you

```
Author: gkh_clanker_t1000 <t1000@kroah.org>
Date:   Fri Jul 10 14:04:17 2026 +0200

    tty: ipwireless: fix use-after-free from deferred rx work outliving tty/network

    ipwireless_detach() frees the tty layer (ipwireless_tty_free()) and the
    network layer (ipwireless_network_free()) before calling
    ipwireless_hardware_free(). But interrupt-driven rx processing is
    deferred through a tasklet and a workqueue item (hw->work_rx,
    scheduled from the tasklet via do_receive_packet()/queue_received_packet()),
    and nothing stopped or waited for that deferred work before the tty and
    network structures it delivers into were freed.

    ipwireless_stop_interrupts() already documents the contract "once this
    function returns, other layers of the driver can be sure they won't
    get any more callbacks", but its implementation only disabled the
    hardware's ability to raise new interrupts (do_close_hardware()) -- it
    never killed the tasklet or flushed already-queued rx work. A tasklet
    scheduled by the last interrupt, or an rx work item it queued, could
    still run after stop_interrupts() returned and reach into the tty/
    network layers via ipwireless_network_packet_received()/tty callbacks.

    The tasklet itself checks hw->shutting_down and bails out early, but
    that only prevents scheduling of *new* rx work; it does not stop a
    tasklet instance already past that check, nor wait for a work item
    already queued.

    Make ipwireless_stop_interrupts() actually honor its documented
    contract by killing the tasklet and flushing hw->work_rx once hardware
    interrupts are disabled. Also call ipwireless_stop_interrupts() from
    ipwireless_detach() before freeing the tty and network layers, instead
    of only relying on it being invoked later inside
    ipwireless_network_free()/ipwireless_hardware_free() -- by then the tty
    layer has already been freed.
```

The bots like to talk

- They want to add too many comments

```
--- a/drivers/tty/ipwireless/hardware.c
+++ b/drivers/tty/ipwireless/hardware.c
@@ -1728,6 +1728,18 @@ void ipwireless_stop_interrupts(struct ipw_hardware *hw)

                /* Prevent the hardware from sending any more interrupts */
                do_close_hardware(hw);
+
+               /*
+                * Make sure no tasklet instance is still running (and can no
+                * longer queue new rx work now that shutting_down is set),
+                * then wait for any rx work already queued to finish. Without
+                * this, a tasklet or work item scheduled just before shutdown
+                * can still run after this function returns and deliver data
+                * into tty/network layers that callers of this function are
+                * about to tear down.
+                */
+               tasklet_kill(&hw->tasklet);
+               flush_work(&hw->work_rx);
        }
 }
```

How the kernel security team has adapted

- Threat model is starting to be documented

How the kernel security team has adapted

- Threat model is starting to be documented
- Security documentation requires cc: maintainers
- Security documentation requests an initial patch

How the kernel security team has adapted

- Threat model is starting to be documented
- Security documentation requires cc: maintainers
- Security documentation requests an initial patch
- Funding for full-time developer to help security@k.o

Protecting your systems

- No **CAP_NET_ADMIN** for untrusted users
- Disable panic_on_warn

What you can do today

- Ignore the doom marketing

What you can do today

- Ignore the doom marketing
- Run local models internally

What you can do today

- Ignore the doom marketing
- Run local models internally
- NEVER upload any non-public information

What you can do today

- Ignore the doom marketing
- Run local models internally
- NEVER upload any non-public information
- Fix the bugs you find today

What you can do today

- Ignore the doom marketing
- Run local models internally
- NEVER upload any non-public information
- Fix the bugs you find today
- Ignore the “model of tomorrow”

Mandatory Manager Reading

- Securing Open Source in the Age of AI

  https://openssf.org/resources/securing-open-source-in-the-age-of-ai-a-practical-guide/

How to do this yourself

- Kernel LLM review prompts

`github.com/masoncl/review-prompts.git`

How to do this yourself

- Get an “agent” (a nanny for the dumb LLM)
  - pi — pi.dev
  - opencode — opencode.ai
  - goose — goose-docs.ai/
  - kres — github.com/masoncl/kres

How to do this yourself

- Get a local model
  - Huggingface `huggingface.co/`
- Get an inference engine
  - Llama.cpp `llama.app/`
  - Ramalama `github.com/containers/ramalama`

How to do this yourself

- Real world example from haproxy maintainer

wtarreau.blogspot.com/2026/05/find-bugs-in-your-code-using-opencode.html

It’s going to be a rough 18 months...