🐧 Kernel Debug Zoo

Workshop Edition

Nandor Kracser

https://github.com/bonifaido/kernel-debug-zoo

🔍 What This Workshop Covers

- Why kernel debugging features matter
- Why you should test against different kernel versions
- Core tools: `lockdep`, `KASAN`, `KMEMLEAK`, `KFENCE`, `SLUB_DEBUG`
- How to trigger them with demo modules
- How Rust could help?

🧠 What Is Kernel Debugging?

- Debugging the **Linux kernel** is very different from userspace:
  - No full stack traces by default
  - Panics can crash the system
  - Subtle bugs (e.g., race conditions) hard to reproduce
- Needs special tooling and awareness (out of the box)

🎯 Goal: Add kernel debbuging tools to your everyday toolbox

🛡️ The 2024 CrowdStrike Kernel Panic

- CrowdStrike Falcon Sensor triggered a **kernel panic** on many Linux systems in April 2024
- Cause: faulty eBPF hook in a widely deployed version
- Broke critical infrastructure — DNS, mail, auth, etc.

🧠 Lessons:

- Always **test kernel code across versions**
- ✅ Validate against all upstream kernels
- ❌ Don’t test only the “happy path”

✅ Takeaway: kernel-space bugs scale globally.

🔐 `lockdep`

Lock Dependency Validator

- Detects incorrect lock ordering
- Flags possible deadlocks
- Works best when locks are used inconsistently

> WARNING: possible circular locking dependency detected

✅ Usually enabled
🛠️ Boot param: `lockdep`

🔥 `KASAN`

Kernel Address Sanitizer

- Detects:
  - Use-after-free
  - Out-of-bounds accesses
- Reports with stack trace

🛠️ Boot params: `kasan=on kasan.multi_shot=1`

⚙️ Needs: `CONFIG_KASAN=y`

🧊 `KFENCE` — Kernel Electric Fence

- Lightweight, always-on memory bug detector
- Similar to KASAN in term os features

🧪 Tradeoffs:

- Lower accuracy than KASAN
- Near-zero performance overhead
- Safe for **production systems**

🛠️ Enable: `kfence.sample_interval=100`

✅ Perfect for long-running systems & catching rare bugs

🧼 `KMEMLEAK`

Memory Leak Detector

- Scans memory for unreachable allocations
- Reports leaks via:

/sys/kernel/debug/kmemleak

🛠️ Boot param: `kmemleak=on`

Enable with: `CONFIG_DEBUG_KMEMLEAK`

🧨 `SLUB` Debugging

Heap Hardening

- Detects:
  - Redzone overflows
  - Double frees
  - Use-after-free
- Works with the `kmalloc()` family of functions
- KASAN provides these features

🛠️ Boot param: `slub_debug=PU`

💣 Page Poisoning

- Catches access to freed pages
- Poisons memory (e.g., with `0xaa`)
- Triggers Oops on access

🛠️ Boot param: `page_poison=1`

Needs: `CONFIG_PAGE_POISONING`

⚔️ Race Condition Demo

- 2 threads update shared var without lock
- Random outputs = race
- `dmesg` might not show it unless it breaks something
- Add manual checks or log divergence

✅ Summary Table

Feature	Catches	Boot Param	Kernel Config
`lockdep`	Deadlocks	`lockdep`	`CONFIG_LOCKDEP`
`KASAN`	UAF, OOB	`kasan=on`	`CONFIG_KASAN`
`KMEMLEAK`	Leaks	`kmemleak=on`	`CONFIG_DEBUG_KMEMLEAK`
`SLUB`	Heap overflows	`slub_debug=PU`	`CONFIG_SLUB_DEBUG_ON`
Page Poison	Freed-page access	`page_poison=1`	`CONFIG_PAGE_POISONING`

🧪 Tips for Demos

- Trigger bugs safely: prefer VMs
- Run `dmesg -w` while testing
- Use `pr_info()` or `trace_printk()`
- Clean up modules: `rmmod`

🛠️ Enabling These Features

Option 1: Use a Prebuilt Debug Kernel

- Fedora/CentOS:
  sudo dnf install kernel-debug sudo grubby --set-default /boot/vmlinuz-<debug-version>
- Comes with most debug configs: `KASAN`, `lockdep`, `SLUB_DEBUG`, etc.

Option 2: Build a Custom Kernel

- Set config options in `make menuconfig`:
- `CONFIG_KASAN=y`
- `CONFIG_DEBUG_KMEMLEAK=y`
- `CONFIG_LOCKDEP_SUPPORT=y`
- `CONFIG_SLUB_DEBUG_ON=y`
- `CONFIG_PAGE_POISONING=y`
- Add boot params in GRUB:
  kasan=on slub_debug=PU kmemleak=on page_poison=1

🦀 Can Rust Help?

Yes — a lot.

- Rust prevents **entire classes** of memory bugs:
- Use-after-free
- Buffer overflows
- Null pointer deref
- Enforced at compile-time

🔐 Rust + Kernel = Safer Drivers

- Rust kernel modules have:
- **Ownership model** for memory safety
- Safe concurrency via `Send` / `Sync`
- No `unsafe` code unless explicitly declared

💡 Still early-stage, but promising for:

- Drivers
- Filesystems
- Net modules

Rust vs C (for Kernel Safety)

Bug Type	C Kernel	Rust Kernel
Use-after-free	✅	🚫 (safe by default)
Buffer overflow	✅	🚫
Null deref	✅	🚫
Data races	✅	🚫 (unless `unsafe`)

💬 Questions?

Let’s go break stuff. 😈