Kernel Panic

km@KM-BBB:~/install$ sudo insmod panic.ko

[sudo] password for km:

[80.310125] panic: no symbol version for module_layout

[80.315547] panic: loading out-of-tree module taints kernel.

[80.321259] panic: module license 'GPLV2' taints kernel.

[80.326772] Disabling lock debugging due to kernel taint

[80.336669] pid:546 comm:insmod

[80.339879] Unable to handle kernel NULL pointer dereference at virtual address 00000000

[80.348221] pgd = f1b3e46e

[80.350948] [00000000] *pgd=00000000

[80.354617] Internal error: Oops: 5 [#1] SMP ARM

[80.359265] Modules linked in: panic(PO+) usb_f_acm u_serial usb_f_ncm usb_f_rndis u_ether libcomposite evdev musb_dsps musb_hdrc usbcore cppi41 ti_am335x_adc kfifo_buf industrialio pm33xx wkup_m3_ipc wkup_m3_rproc remoteproc virtio virtio_ring ti_emif_sram tps65218_pwrbutton omap_rng rng_core rtc_ds1307 hwmon at24 omap_mailbox musb_am335x rtc_omap omap_wdt ti_am335x_tscadc leds_gpio led_class cpufreq_dt autofs4

[80.396005] CPU: 0 PID: 546 Comm: insmod Tainted: P O 4.19.94-gbce566dcf-dirty #4

[80.404831] Hardware name: Generic AM33XX (Flattened Device Tree)

[80.410970] PC is at panic2_init+0x34/0x50 [panic]

[80.415790] LR is at panic2_init+0x28/0x50 [panic]

[80.420605] pc : [<bf238034>] lr : [<bf238028>] psr: 600f0013

[80.426902] sp : da873dc0 ip : 00000000 fp : c0e08948

[80.432151] r10: bf23a000 r9 : 00000028 r8 : 00000000

[80.437401] r7 : bf238000 r6 : c0e08948 r5 : c0e08974 r4 : c0ec7100

[80.443959] r3 : 00000000 r2 : 4a99e67f r1 : 00000000 r0 : bf239050

[80.450521] Flags: nZCv IRQs on FIQs on Mode SVC_32 ISA ARM Segment none

[80.457691] Control: 10c5387d Table: 9ac40019 DAC: 00000051

[80.463465] Process insmod (pid: 546, stack limit = 0x38b2db9b)

[80.469414] Stack: (0xda873dc0 to 0xda874000)

[80.473800] 3dc0: c0ec7100 c0102fd8 006000c0 006000c0 00000001 4a99e67f 00000028 bf23a000

[80.482021] 3de0: 006000c0 006000c0 dd3b6740 006000c0 006000c0 c163e7dc de0000c0 c02d330c

[80.490241] 3e00: c0e08974 daca8640 c02c5b04 0000000c daca8640 4a99e67f bf23a000 c0ec77b8

[80.498462] 3e20: dd3b6740 00000001 dacaa870 00000028 bf23a000 c01f37b8 da873f38 c0ec77b8

[80.506683] 3e40: da873f38 c0ec77b8 c0e08974 c01f5c40 bf23a00c 00007fff bf23a000 c01f29e0

[80.514903] 3e60: 00000001 bf23a048 bf23a00c bf23a154 bf23a170 bf23a000 bf23a22c c0a06208

[80.523123] 3e80: da873f2c c0bcf508 c02f0001 00000000 c0c70628 c0c6187c 00000000 00000000

[80.531344] 3ea0: 00000000 00000000 00000000 00000000 6e72656b 00006c65 00000000 00000000

[80.539564] 3ec0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

[80.547784] 3ee0: 00000000 00000000 00000000 4a99e67f 7fffffff c0e08948 00000000 00000003

[80.556005] 3f00: 004de7e0 7fffffff 00000000 0000017b 00000000 c01f61c8 7fffffff 00000000

[80.564226] 3f20: 00000003 c019ea0c da873f4c e2c49000 00004aec 00000000 e2c4915e e2c49240

[80.572446] 3f40: e2c49000 00004aec e2c4d4ac e2c4d328 e2c4d02c 00003000 00003130 00000000

[80.580666] 3f60: 00000000 00000000 000004fc 00000025 00000026 0000000d 0000000b 00000008

[80.588887] 3f80: 00000000 4a99e67f 00000028 4e831300 00000000 00000000 0000017b c01011c4

[80.597107] 3fa0: da872000 c0101000 4e831300 00000000 00000003 004de7e0 00000000 bebdbc68

[80.605327] 3fc0: 4e831300 00000000 00000000 0000017b 004f4590 00000000 bebdbde8 00000000

[80.613547] 3fe0: bebdbc18 bebdbc08 004d6e41 b6cfad92 40030030 00000003 00000000 00000000

[80.621820] [<bf238034>] (panic2_init [panic]) from [<c0102fd8>] (do_one_initcall+0x80/0x318)

[80.630401] [<c0102fd8>] (do_one_initcall) from [<c01f37b8>] (do_init_module+0x5c/0x1f8)

[80.638537] [<c01f37b8>] (do_init_module) from [<c01f5c40>] (load_module+0x2284/0x25a4)

[80.646583] [<c01f5c40>] (load_module) from [<c01f61c8>] (sys_finit_module+0xbc/0xdc)

[80.654455] [<c01f61c8>] (sys_finit_module) from [<c0101000>] (ret_fast_syscall+0x0/0x28)

[80.662671] Exception stack(0xda873fa8 to 0xda873ff0)

[80.667752] 3fa0: 4e831300 00000000 00000003 004de7e0 00000000 bebdbc68

[80.675972] 3fc0: 4e831300 00000000 00000000 0000017b 004f4590 00000000 bebdbde8 00000000

[80.684190] 3fe0: bebdbc18 bebdbc08 004d6e41 b6cfad92

[80.689273] Code: eb3de61e e59f3018 e59f0018 e5933000 (e5931000)

[80.695553] ---[end trace 654b5c69b6139492]---

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.354617] Internal error: Oops: 5 [#1] SMP ARM

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.463465] Process insmod (pid: 546, stack limit = 0x38b2db9b)

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.469414] Stack: (0xda873dc0 to 0xda874000)

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.473800] 3dc0: c0ec7100 c0102fd8 006000c0 006000c0 00000001 4a99e67f 00000028 bf23a000

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.482021] 3de0: 006000c0 006000c0 dd3b6740 006000c0 006000c0 c163e7dc de0000c0 c02d330c

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.490241] 3e00: c0e08974 daca8640 c02c5b04 0000000c daca8640 4a99e67f bf23a000 c0ec77b8

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.498462] 3e20: dd3b6740 00000001 dacaa870 00000028 bf23a000 c01f37b8 da873f38 c0ec77b8

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.506683] 3e40: da873f38 c0ec77b8 c0e08974 c01f5c40 bf23a00c 00007fff bf23a000 c01f29e0

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.514903] 3e60: 00000001 bf23a048 bf23a00c bf23a154 bf23a170 bf23a000 bf23a22c c0a06208

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.523123] 3e80: da873f2c c0bcf508 c02f0001 00000000 c0c70628 c0c6187c 00000000 00000000

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.531344] 3ea0: 00000000 00000000 00000000 00000000 6e72656b 00006c65 00000000 00000000

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.539564] 3ec0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.547784] 3ee0: 00000000 00000000 00000000 4a99e67f 7fffffff c0e08948 00000000 00000003

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.556005] 3f00: 004de7e0 7fffffff 00000000 0000017b 00000000 c01f61c8 7fffffff 00000000

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.564226] 3f20: 00000003 c019ea0c da873f4c e2c49000 00004aec 00000000 e2c4915e e2c49240

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.572446] 3f40: e2c49000 00004aec e2c4d4ac e2c4d328 e2c4d02c 00003000 00003130 00000000

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.580666] 3f60: 00000000 00000000 000004fc 00000025 00000026 0000000d 0000000b 00000008

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.588887] 3f80: 00000000 4a99e67f 00000028 4e831300 00000000 00000000 0000017b c01011c4

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.597107] 3fa0: da872000 c0101000 4e831300 00000000 00000003 004de7e0 00000000 bebdbc68

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.605327] 3fc0: 4e831300 00000000 00000000 0000017b 004f4590 00000000 bebdbde8 00000000

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.613547] 3fe0: bebdbc18 bebdbc08 004d6e41 b6cfad92 40030030 00000003 00000000 00000000

Message from syslogd@KM-BBB at Jul 21 08:54:21 ...

kernel:[80.689273] Code: eb3de61e e59f3018 e59f0018 e5933000 (e5931000)

Segmentation fault

---

1) Quick summary (one sentence)

Your module (panic.ko) caused a **kernel NULL-pointer dereference** during its init routine panic2_init. The kernel printed an Oops with the program counter (PC) inside panic2_init — that's how you know where it crashed.

---

2) Important log lines — plain English

- [80.310125] panic: no symbol version for module_layout
  The module was built with a different symbol-versioning configuration than the running kernel. Not fatal by itself but a sign the module wasn't built against the exact running kernel headers/config.
- [80.315547] panic: loading out-of-tree module taints kernel.
  The kernel marks itself "tainted" because an out-of-tree module (not from the official kernel build) was loaded. This affects support/diagnostics (kernel devs may refuse support for tainted kernels).
- [80.321259] panic: module license 'GPLV2' taints kernel.
  The MODULE_LICENSE() string in your module is GPLV2 (nonstandard). The kernel expects GPL/GPL v2 etc. This also causes taint. (Fix: use MODULE_LICENSE("GPL");.)
- [80.326772] Disabling lock debugging due to kernel taint
  Because the kernel is tainted, some runtime debugging (lockdep) is disabled.

- Unable to handle kernel NULL pointer dereference at virtual address 00000000
  The kernel attempted to read or write memory at address 0x0 (NULL) — a classic null-pointer dereference in kernel space.
- pgd = f1b3e46e and [00000000] *pgd=00000000
  The page-global directory entry for address 0x0 is empty — page mapping absent for that address. That's why dereferencing 0x0 faults.
- Internal error: Oops: 5 [#1] SMP ARM
  This is an Oops (recoverable kernel error). The CPU/arch is ARM.
- Modules linked in: panic(PO+) ...
  Lists loaded modules; panic(PO+) is your module and shows taint flags.
- PC is at panic2_init+0x34/0x50 [panic]
  Key line. The instruction pointer (PC) is inside function panic2_init at offset 0x34 within that function (function size 0x50). So the crash occurred in panic2_init.
- pc : [<bf238034>] lr : [<bf238028>]
  pc is the runtime virtual address of the instruction that faulted. lr (link register) is the return address.
- r0..r10, sp, ip, fp registers shown next
  These are the CPU register values at the time of crash. Notably r3 : 00000000 — that indicates the code was using register r3 as a pointer, and it contained 0 (NULL).
- Code: eb3de61e e59f3018 e59f0018 e5933000 (e5931000)
  This is raw ARM machine code bytes and the current instruction in parentheses: e5931000. That instruction is an LDR (load) using r3 as base (load from [r3]). Since r3 is 0, LDR r1, [r3] caused the fault.
- Stack trace lines:
- [<bf238034>] (panic2_init [panic]) from [<c0102fd8>] (do_one_initcall+0x80/0x318)
- ...

This shows the call chain: panic2_init was called by do_one_initcall (the kernel initcall machinery that runs module init functions). The hex values in [] are addresses for each frame.

- ---[end trace ...]---
  End of Oops trace.
- Final Segmentation fault printed to your shell:
  Because the kernel oops happened while insmod was running, the insmod process got an error / died (user-space saw a segfault). Kernel Oops often disrupt the user process that triggered the action.

---

3) Why it crashed (technical but clear)

Inside your module's init function panic2_init, the code executed a load instruction that used a register containing 0x0 (NULL) as the memory base. The instruction e5931000 decodes to something like ldr r1, [r3] — read memory from the address in r3. Since r3 == 0x0, the CPU tried to read address 0x0 and that's an invalid kernel address → page fault → Oops.

This is exactly what happens if your init code does something like:

int *p = NULL;

int val = *p; // deref NULL — crash

(or you dereference a structure pointer that wasn't initialized).

---

4) How to map that crash to the exact source line (step-by-step commands)

You already have the active PC: 0xbf238034. The module base address is usually shown in /proc/modules. The process:

A. Get the module base address

on the BeagleBone where crash happened

grep '^panic ' /proc/modules

sample output format: panic 16384 0 - Live 0xbf238000

the last hex is the load address (module base)

B. compute the offset of PC into the module
Replace 0xbf238034 below with the pc value from your dmesg and the base from /proc/modules:

pc=0xbf238034

base=$(awk '/^panic /{print $6}' /proc/modules)

compute numeric offset

offset=$((pc - base))

printf "pc=%s base=%s offset=%#x\n" $pc $base $offset

In your Oops the offset will come out around 0x34 (because kernel already printed panic2_init+0x34).

C. Map offset to source file/line using addr2line
Use the same cross-toolchain you used to build (or the native addr2line if you built natively):

on the build machine that has panic.ko and addr2line

arm-linux-gnueabihf-addr2line -f -C -e panic.ko 0x34

or if offset variable available:

arm-linux-gnueabihf-addr2line -f -C -e panic.ko $offset

-f -C prints function and demangled names. The output will be panic.c:LINE — that is the exact source line.

D. Alternative: use objdump or gdb

disassemble the module to inspect the function

arm-linux-gnueabihf-objdump -d panic.ko | sed -n '/<panic2_init>/,/<panic2_init>/+40p'

or open in gdb (shows source if built with -g)

arm-linux-gnueabihf-gdb panic.ko

(gdb) list *0x34

(gdb) info line *0x34

Notes:

- addr2line / objdump must be run against the same panic.ko binary you inserted (the one built with the same symbols). If you compiled panic.ko on a host and copied to the BBB, run the mapping on the host where the binary is.
- If you used the absolute runtime address (e.g. 0xbf238034) directly with addr2line, it won't help — you must use the **offset inside the module** (pc - module_base), or use add-symbol-file in gdb giving the runtime base.

---

5) What the panic2_init+0x34/0x50 notation means

- panic2_init — function name.
- +0x34 — offset (in bytes) from start of panic2_init where the fault happened.
- /0x50 — the total size of panic2_init function is 0x50 bytes.

So the crash is within the function body, 52 bytes from its start.

---

6) Build / taint / symbol version problems — what they mean and fixes

- **no symbol version for module_layout**
  This means the kernel expects symbol versioning (CONFIG_MODVERSIONS) or the module was built against different kernel headers. **Fix**: build the module against /lib/modules/$(uname -r)/build (the kernel build directory for the running kernel). Example Makefile invocation:

- make -C /lib/modules/$(uname -r)/build M=$PWD modules

That ensures vermagic and symbol-versioning align.

- **loading out-of-tree module taints kernel** & **module license 'GPLV2' taints kernel**
  Kernel marks itself tainted because the module came from outside the kernel tree and the license string isn't recognized. To avoid that use the standard license macro:
- MODULE_LICENSE("GPL");

(and rebuild). Note: it's only a support/diagnostics issue; taint does not necessarily cause the crash.

- **Disabling lock debugging due to kernel taint**
  If you want lock debugging (lockdep), run a non-tainted kernel (i.e., load only in-tree modules or use a recognized license).

---

7) How to avoid this class of crash in future (practical tips)

- Initialize pointers before use. Never dereference pointers you haven't set.
- Check return values of allocation functions (kmalloc, dev_get_drvdata, etc.) before use.
- Build your module with debug symbols: add EXTRA_CFLAGS += -g in module Makefile or pass CFLAGS so addr2line/gdb show real source lines.
- Build modules with the exact kernel source/config used at runtime.
- Use printk() early in init to confirm pointers are non-NULL before deref.
- Use sparse or smatch static checks for obvious bugs.

---

8) Concrete checklist you can run now on the BBB

1. Show dmesg around the oops:
2. dmesg -T | sed -n '1,200p'
3. Get module base:
4. grep '^panic ' /proc/modules
5. Compute offset & map to source (example using the values you have):
6. # replace with your PC from dmesg if different
7. pc=0xbf238034
8. base=$(awk '/^panic /{print $6}' /proc/modules)

9. offset=$((pc - base))
10. printf "pc=%s base=%s offset=%#x\n" $pc $base $offset
11.
12. # run on build machine with the same panic.ko
13. arm-linux-gnueabihf-addr2line -f -C -e panic.ko $offset
14. If you get a file:line from addr2line, open the source file and inspect the line (looking for a NULL dereference) and add checks.

---

9) Example: your crash decoded (based on the trace)

- r3 == 0x0 and the executed instruction was LDR r1, [r3] → code attempted *(r3) where r3 was NULL. That corresponds to a line in panic.c (or whatever your module file is called) where you dereference a pointer that wasn't initialized. The addr2line step will confirm the exact source line.

---

10) Want me to do anything next?

If you want, tell me one of the following and I'll produce it immediately (no waiting):

- I can produce the **exact command sequence** you should run and a small Makefile snippet to build panic.ko correctly with -g and correct vermagic.
- If you upload the panic.ko binary (or paste its objdump -t panic.ko / nm panic.ko output), I can show the objdump/symbol lines and point to the failing instruction.
- Or paste the panic.c source and I'll point out the offending line and a safe fix.

Which one do you want me to do now?