Kernel Debugging

# Tainted: X

Message at kernel panic:

CPU: 4 PID: 8 Comm: kworker/u12:0 Tainted: G

File : kernel/panic.c

const struct taint\_flag taint\_flags[TAINT\_FLAGS\_COUNT] = {

[ TAINT\_PROPRIETARY\_MODULE ] = { 'P', 'G', true },

[ TAINT\_FORCED\_MODULE ] = { 'F', ' ', true },

[ TAINT\_CPU\_OUT\_OF\_SPEC ] = { 'S', ' ', false },

[ TAINT\_FORCED\_RMMOD ] = { 'R', ' ', false },

[ TAINT\_MACHINE\_CHECK ] = { 'M', ' ', false },

[ TAINT\_BAD\_PAGE ] = { 'B', ' ', false },

[ TAINT\_USER ] = { 'U', ' ', false },

[ TAINT\_DIE ] = { 'D', ' ', false },

[ TAINT\_OVERRIDDEN\_ACPI\_TABLE ] = { 'A', ' ', false },

[ TAINT\_WARN ] = { 'W', ' ', false },

[ TAINT\_CRAP ] = { 'C', ' ', true },

[ TAINT\_FIRMWARE\_WORKAROUND ] = { 'I', ' ', false },

[ TAINT\_OOT\_MODULE ] = { 'O', ' ', true },

[ TAINT\_UNSIGNED\_MODULE ] = { 'E', ' ', true },

[ TAINT\_SOFTLOCKUP ] = { 'L', ' ', false },

[ TAINT\_LIVEPATCH ] = { 'K', ' ', true },

[ TAINT\_AUX ] = { 'X', ' ', true },

[ TAINT\_RANDSTRUCT ] = { 'T', ' ', true },

};

## **Taint flags**

The tainted status of the kernel not only indicates whether or not the kernel has been tainted but also indicates what type(s) of event caused the kernel to be marked as tainted. This information is encoded through single-character flags in the string following "Tainted:" in a kernel error message.

**P**: A module with a Proprietary license has been loaded, i.e. a module that is not licensed under the GNU General Public License (GPL) or a compatible license. This may indicate that source code for this module is not available to the Linux kernel developers or to SUSE developers.

**G**: The opposite of '**P**': the kernel has been tainted (for a reason indicated by a different flag), but all modules loaded into it were licensed under the GPL or a license compatible with the GPL.

**F**: A module was loaded using the Force option "-f" of insmod or modprobe, which caused a sanity check of the versioning information from the module (if present) to be skipped.

**R**: A module which was in use or was not designed to be removed has been forcefully Removed from the running kernel using the force option "-f" of rmmod.

**S**: The Linux kernel is running with Symmetric MultiProcessor support (SMP), but the CPUs in the system are not designed or certified for SMP use.

**M**: A Machine Check Exception (MCE) has been raised while the kernel was running. MCEs are triggered by the hardware to indicate a hardware related problem, for example the CPU's temperature exceeding a threshold or a memory bank signaling an uncorrectable error.

**B**: A process has been found in a Bad page state, indicating a corruption of the virtual memory subsystem, possibly caused by malfunctioning RAM or cache memory.  
  
**U**: User or user application specifically requested that the Tainted flag be set, ' ' otherwise.  
  
**D**: Kernel has Died recently, i.e. there was an OOPS or BUG.  
  
**W**: A Warning has previously been issued by the kernel.  
  
**C**: A staging driver has been loaded.  
  
**A**: ACPI table has been overriden [From SLES11 SP1 onwards]  
  
**I**: Kernel is working around a severe bug in the platform firmware [From SLES11 SP2 onwards]

**O**: An Out-of-tree module has been loaded [From SLES12 SP0 onwards]

**L**: A Soft Lockup has previously occured on the system [From SLES12 SP2 onwards]

**K**: The Kernel has been live patched [From SLES12 SP2 onwards]

The taint flags above are implemented in the standard Linux kernel and indicate that the information provided in kernel error messages is not necessarily to be trusted.

Decoding Tainted kernel: -

https://docs.kernel.org/admin-guide/tainted-kernels.html

# AARCH64 Exception level: -

<https://krinkinmu.github.io/2021/01/04/aarch64-exception-levels.html>

<https://medium.com/@om.nara/aarch64-exception-levels-60d3a74280e6#:~:text=EL2%20provides%20only%20Non%2DSecure,specific%20hardware%20implementation%20of%20AArch64>.

# Interpretation of below log extract: -

2 3 1:[ 39.813684][ T187] Mem abort info:

2 3 1:[ 39.814827][ T187] ESR = 0x86000004

2 3 1:[ 39.814975][ T187] EC = 0x21: IABT (current EL), IL = 32 bits

2 3 1:[ 39.816948][ T187] SET = 0, FnV = 0

2 3 1:[ 39.817126][ T187] EA = 0, S1PTW = 0

2 3 1:[ 39.817245][ T187] FSC = 0x04: level 0 translation fault

T187 - > Thread ID

In above, using “2 3 1”, it indicates that VM ID is 3 i.e. Android. One of remaining 1, 2 represents CPU ID.

# Use-After-Free Bug: -

UAF bugs occur when a program continues to access a memory location after the memory has been freed or deallocated.

# KASAN

References:

1. <https://www.starlab.io/blog/kasan-what-is-it-how-does-it-work-and-what-are-the-strange-numbers-at-the-end>

Kernel debugging config options● KASAN – Find out of bounds accesses and use-after-free bugs using  
shadow memory (~valgrind) or sw/hw tags  
● GENERIC - Instrument each access to check shadow memory  
● Cost is 1/8 memory and 3x slower performance, needs new enough GCC or Clang  
● SW\_TAGS – embeds tags to pointers, checks by instrumentation  
● Only slab and page allocations, arm64 with Top Byte Ignore, Clang, 1/16 memory  
● HW\_TAGS – arm64 with Memory Tagging Extension, checks by hardware  
● Also slab and page allocations, only reports first bug, then disables itself