Tainted kernels

The kernel will mark itself as 'tainted' when something occurs that might be relevant later when investigating problems. Don't worry too much about this, most of the time it's not a problem to run a tainted kernel; the information is mainly of interest once someone wants to investigate some problem, as its real cause might be the event that got the kernel tainted. That's why bug reports from tainted kernels will often be ignored by developers, hence try to reproduce problems with an untainted kernel.

Note the kernel will remain tainted even after you undo what caused the taint (i.e. unload a proprietary kernel module), to indicate the kernel remains not trustworthy. That's also why the kernel will print the tainted state when it notices an internal problem (a 'kernel bug'), a recoverable error ('kernel oops') or a non-recoverable error ('kernel panic') and writes debug information about this to the logs `dmesg` outputs. It's also possible to check the tainted state at runtime through a file in `/proc/`.

Tainted flag in bugs, oops or panics messages

You find the tainted state near the top in a line starting with 'CPU:'; if or why the kernel was tainted is shown after the Process ID ('PID:') and a shortened name of the command ('Comm:') that triggered the event:

```
BUG: unable to handle kernel NULL pointer dereference at 0000000000000000
Oops: 0002 [#1] SMP PTI
CPU: 0 PID: 4424 Comm: insmod Tainted: P        W  O      4.20.0-0.rc6.fc30 #1
Hardware name: Red Hat KVM, BIOS 0.5.1 01/01/2011
RIP: 0010:my_oops_init+0x13/0x1000 [kpanic]
[...]
```

You'll find a 'Not tainted: ' there if the kernel was not tainted at the time of the event; if it was, then it will print 'Tainted: ' and characters either letters or blanks. In above example it looks like this:

```
Tainted: P        W  O
```

The meaning of those characters is explained in the table below. In this case the kernel got tainted earlier because a proprietary Module (`P`) was loaded, a warning occurred (`W`), and an externally-built module was loaded (`O`). To decode other letters use the table below.

Decoding tainted state at runtime

At runtime, you can query the tainted state by reading `cat /proc/sys/kernel/tainted`. If that returns `0`, the kernel is not tainted; any other number indicates the reasons why it is. The easiest way to decode that number is the script `tools/debugging/kernel-chktaint`, which your distribution might ship as part of a package called `linux-tools` or `kernel-tools`; if it doesn't you can download the script from git.kernel.org and execute it with `sh kernel-chktaint`, which would print something like this on the machine that had the statements in the logs that were quoted earlier:

```
Kernel is Tainted for following reasons:
 * Proprietary module was loaded (#0)
 * Kernel issued warning (#9)
 * Externally-built ('out-of-tree') module was loaded  (#12)
See Documentation/admin-guide/tainted-kernels.rst in the Linux kernel or
 https://www.kernel.org/doc/html/latest/admin-guide/tainted-kernels.html for
 a more details explanation of the various taint flags.
Raw taint value as int/string: 4609/'P        W  O     '
```

You can try to decode the number yourself. That's easy if there was only one reason that got your kernel tainted, as in this case you can find the number with the table below. If there were multiple reasons you need to decode the number, as it is a bitfield, where each bit indicates the absence or presence of a particular type of taint. It's best to leave that to the aforementioned script, but if you need something quick you can use this shell command to check which bits are set:

```
$ for i in $(seq 18); do echo $(($i-1)) $(($(cat /proc/sys/kernel/tainted)>>($i-1)&1));done
```

Table for decoding tainted state

Bit	Log	Number	Reason that got the kernel tainted
0	G/P	1	proprietary module was loaded
1	_/F	2	module was force loaded
2	_/S	4	kernel running on an out of specification system
3	_/R	8	module was force unloaded
4	_/M	16	processor reported a Machine Check Exception (MCE)
5	_/B	32	bad page referenced or some unexpected page flags
6	_/U	64	taint requested by userspace application
7	_/D	128	kernel died recently, i.e. there was an OOPS or BUG

Bit	Log	Number	Reason that got the kernel tainted
8	_/A	256	ACPI table overridden by user
9	_/W	512	kernel issued warning
10	_/C	1024	staging driver was loaded
11	_/I	2048	workaround for bug in platform firmware applied
12	_/O	4096	externally-built ("out-of-tree") module was loaded
13	_/E	8192	unsigned module was loaded
14	_/L	16384	soft lockup occurred
15	_/K	32768	kernel has been live patched
16	_/X	65536	auxiliary taint, defined for and used by distros
17	_/T	131072	kernel was built with the struct randomization plugin

Note: The character _ is representing a blank in this table to make reading easier.

More detailed explanation for tainting

0. `G` if all modules loaded have a GPL or compatible license, `P` if any proprietary module has been loaded. Modules without a MODULE_LICENSE or with a MODULE_LICENSE that is not recognised by insmod as GPL compatible are assumed to be proprietary.
1. `F` if any module was force loaded by `insmod -f`, `' '` if all modules were loaded normally.
2. `S` if the kernel is running on a processor or system that is out of specification: hardware has been put into an unsupported configuration, therefore proper execution cannot be guaranteed. Kernel will be tainted if, for example:
    - on x86: PAE is forced through forcepae on intel CPUs (such as Pentium M) which do not report PAE but may have a functional implementation, an SMP kernel is running on non officially capable SMP Athlon CPUs, MSRs are being poked at from userspace.
    - on arm: kernel running on certain CPUs (such as Keystone 2) without having certain kernel features enabled.
    - on arm64: there are mismatched hardware features between CPUs, the bootloader has booted CPUs in different modes.
    - certain drivers are being used on non supported architectures (such as scsi/snic on something else than x86_64, scsi/ips on non x86/x86_64/itanium, have broken firmware settings for the irqchip/irq-gic on arm64 ...).
3. `R` if a module was force unloaded by `rmmod -f`, `' '` if all modules were unloaded normally.
4. `M` if any processor has reported a Machine Check Exception, `' '` if no Machine Check Exceptions have occurred.
5. `B` If a page-release function has found a bad page reference or some unexpected page flags. This indicates a hardware problem or a kernel bug; there should be other information in the log indicating why this tainting occurred.
6. `U` if a user or user application specifically requested that the Tainted flag be set, `' '` otherwise.
7. `D` if the kernel has died recently, i.e. there was an OOPS or BUG.
8. `A` if an ACPI table has been overridden.
9. `W` if a warning has previously been issued by the kernel. (Though some warnings may set more specific taint flags.)
10. `C` if a staging driver has been loaded.
11. `I` if the kernel is working around a severe bug in the platform firmware (BIOS or similar).
12. `O` if an externally-built ("out-of-tree") module has been loaded.
13. `E` if an unsigned module has been loaded in a kernel supporting module signature.
14. `L` if a soft lockup has previously occurred on the system.
15. `K` if the kernel has been live patched.
16. `X` Auxiliary taint, defined for and used by Linux distributors.
17. `T` Kernel was build with the randstruct plugin, which can intentionally produce extremely unusual kernel structure layouts (even performance pathological ones), which is important to know when debugging. Set at build time.