Loadable kernel module

By Contributors to Wikimedia projects

Published: 2003-11-18 · Archived: 2026-04-05 17:53:03 UTC

From Wikipedia, the free encyclopedia

A **loadable kernel module** (**LKM**) is an executable library that extends the capabilities of a running kernel, or so-called *base kernel*, of an operating system. LKMs are typically used to add support for new hardware (as device drivers) and/or filesystems, or for adding system calls. When the functionality provided by an LKM is no longer required, it can be unloaded in order to free memory and other resources.

Most current Unix-like systems and Windows support loadable kernel modules but with different names, such as **kernel loadable module** (kld) in FreeBSD, **kernel extension** (kext) in macOS (although support for third-party modules is being dropped[1]),[2] **kernel extension module** in AIX, **dynamically loadable kernel module** in HP-UX,[3] **kernel-mode driver** in Windows NT[4] and **downloadable kernel module** (DKM) in VxWorks. They are also known as **kernel loadable module** (KLM), or simply as **kernel module** (KMOD).

Without loadable kernel modules, an operating system would have to include all possible anticipated functionality compiled directly into the base kernel. Much of that functionality would reside in memory without being used, wasting memory [*citation needed*], and would require that users rebuild and reboot the base kernel every time they require new functionality.

One minor criticism of preferring a modular kernel over a static kernel is the so-called *fragmentation penalty*. The base kernel is always unpacked into real contiguous memory by its setup routines; thus, the base kernel code is never fragmented. Once the system is in a state in which modules may be inserted, for example once the filesystems have been mounted that contain the modules, it is likely that any new kernel code insertion will cause the kernel to become fragmented, thereby introducing a minor performance penalty by using more TLB entries, causing more TLB misses.[*citation needed*]

Implementations in different operating systems

[edit]

Loadable kernel modules in Linux are loaded (and unloaded) by the `modprobe` command. They are located in `/lib/modules` or `/usr/lib/modules` and have had the extension `.ko` ("kernel object") since version 2.6 (previous versions used the `.o` extension).[5] The `lsmod` command lists the loaded kernel modules. In emergency cases, when the system fails to boot due to e.g. broken modules, specific modules can be enabled or disabled by modifying the kernel boot parameters list (for example, if using GRUB, by pressing 'e' in the GRUB start menu, then editing the kernel parameter line).

In the opinion of Linux maintainers, LKM are derived works of the kernel[*citation needed*]. The Linux maintainers tolerate the distribution of proprietary modules (such as NVIDIA GPU drivers),[*citation needed*] but allow only GNU General Public License (GPL) modules to merge to kernel tree of mainline Linux kernel.

Loading a proprietary or non-GPL-compatible module will set a 'taint' flag[6][7] in the running kernel—meaning that any problems or bugs experienced will be less likely to be investigated by the maintainers.[8][9] LKMs effectively become part of the running kernel, so can corrupt kernel data structures and produce bugs that may not be able to be investigated if the module is indeed proprietary.

Linuxant controversy

[edit]

In 2004, Linuxant, a consulting company that releases proprietary device drivers as loadable kernel modules, attempted to abuse a null terminator in their `MODULE_LICENSE`, as visible in the following code excerpt:

```
MODULE_LICENSE("GPL\0for files in the \"GPL\" directory; for others, only LICENSE file applies");
```

The string comparison code used by the kernel at the time tried to determine whether the module was GPLed stopped when it reached a null character (`\0`), so it was fooled into thinking that the module was declaring its license to be just "GPL".[10]

Kernel modules for FreeBSD are stored within `/boot/kernel/` for modules distributed with the operating system, or usually `/boot/modules/` for modules installed from FreeBSD ports or FreeBSD packages, or for proprietary or otherwise binary-only modules. FreeBSD kernel modules usually have the extension `.ko`. Once the machine has booted, they may be loaded with the `kldload` command, unloaded with `kldunload`, and listed with `kldstat`. Modules can also be loaded from the loader before the kernel starts, either automatically (through `/boot/loader.conf`) or by hand.

Some loadable kernel modules in macOS can be loaded automatically. Loadable kernel modules can also be loaded by the `kextload` command. They can be listed by the `kextstat` command. Loadable kernel modules are located in bundles with the extension `.kext`. Modules supplied with the operating system are stored in the `/System/Library/Extensions` directory; modules supplied by third parties are in various other directories.

A NetWare kernel module is referred to as a NetWare Loadable Module (NLM). NLMs are inserted into the NetWare kernel by means of the LOAD command, and removed by means of the UNLOAD command; the `modules` command lists currently loaded kernel modules. NLMs may reside in any valid search path assigned on the NetWare server, and they have `.NLM` as the file name extension.

A downloadable kernel module (DKM) type project can be created to generate a ".out" file which can then be loaded to kernel space using "ld" command. This downloadable kernel module can be unloaded using "unld" command.

Solaris has a configurable kernel module load path, which defaults to `/platform/platform-name/kernel /kernel /usr/kernel`. Most kernel modules live in subdirectories under `/kernel`; those not considered necessary to boot the system to the point that init can start are often (but not always) found in `/usr/kernel`. When running a DEBUG kernel build the system actively attempts to unload modules.

Binary compatibility

[edit]

Linux does not provide a stable API or ABI for kernel modules. This means that there are differences in internal structure and function between different kernel versions, which can cause compatibility problems. In an attempt to combat those problems, symbol versioning data is placed within the `.modinfo` section of loadable ELF modules. This versioning information can be compared with that of the running kernel before loading a module; if the versions are incompatible, the module will not be loaded.

Other operating systems, such as Solaris, FreeBSD, macOS, and Windows keep the kernel API and ABI relatively stable, thus avoiding this problem. For example, FreeBSD kernel modules compiled against kernel version 6.0 will work without recompilation on any other FreeBSD 6.x version, e.g. 6.4. However, they are not compatible with other major versions and must be recompiled for use with FreeBSD 7.x, as API and ABI compatibility is maintained only within a branch.

While loadable kernel modules are a convenient method of modifying the running kernel, this can be abused by attackers on a compromised system to prevent detection of their processes or files, allowing them to maintain control over the system. Many rootkits make use of LKMs in this way. Note that, on most operating systems, modules do not help privilege elevation in any way, as elevated privilege is required to load a LKM; they merely make it easier for the attacker to hide the break-in.[11]

Linux allows disabling module loading via sysctl option `/proc/sys/kernel/modules_disabled`.[12][13] An initramfs system may load specific modules needed for a machine at boot and then disable module loading. This makes the security very similar to a monolithic kernel. If an attacker can change the initramfs, they can change the kernel binary.

In OS X Yosemite and later releases, a kernel extension has to be code-signed with a developer certificate that holds a particular "entitlement." Such a developer certificate is only provided by Apple on request and not automatically given to Apple Developer members. This feature, called "kext signing", is enabled by default and it instructs the kernel to stop booting if unsigned kernel extensions are present.[14] In OS X El Capitan and later releases, it is part of System Integrity Protection.

In older versions of macOS, or if kext signing is disabled, a loadable kernel module in a kernel extension bundle can be loaded by non-root users if the OSBundleAllowUserLoad property is set to True in the bundle's property list.[15] However, if any of the files in the bundle, including the executable code file, are not owned by root and group wheel, or are writable by the group or "other", the attempt to load the kernel loadable module will fail.[16]

Kernel modules can optionally have a cryptographic signature ELF section which is verified on load depending on the Verified Boot policy settings. The kernel can enforce that modules are cryptographically signed by a set of

trusted certificates; the list of trusted certificates is held outside of the OS in the ILOM on some SPARC based platforms. Userspace initiated kernel module loading is only possible from the Trusted Path when the system is running with the Immutable Global Zone feature enabled.

- Dynamic link library – Sharable executable library in Windows and OS/2
- NetWare Loadable Module – Novell-compatible computer-readable software
- Shared library – Software library in memory that multiple executables can use at runtime

1. ^ *"Deprecated Kernel Extensions and System Extension Alternatives". Apple Inc. Retrieved 13 March 2021.*
2. ^ *"Kernel Extension Programming Topics: Introduction". Apple Inc. September 1, 2010. Archived from the original on May 4, 2013. Retrieved May 5, 2013.*
3. ^ *"Managing and Developing DynamicallyLoadable Kernel Modules". Hewlett-Packard. June 7, 2001.*
4. ^ *"What Determines When a Driver Is Loaded". Microsoft Developer Network. Microsoft. November 21, 2012. Archived from the original on March 6, 2013. Retrieved May 5, 2013.*
5. ^ *"The Linux Kernel Module Programming Guide, section 2.2 "Compiling Kernel Modules"". Retrieved 2020-10-05.*
6. ^ *Linus Torvalds; et al. (2011-06-21). "Documentation/oops-tracing.txt". kernel.org. Archived from the original on 2011-10-02. Retrieved 2011-10-03.*
7. ^ *"Tainted kernels". The Linux kernel user's and administrator's guide.*
8. ^ *Jonathan Corbet (2006-03-24). "Tainting from user space". LWN.net. Archived from the original on 2011-11-16. Retrieved 2011-10-03.*
9. ^ *"Novell support documentation: Tainted kernel". 2007-07-26. Retrieved 2011-10-03.*
10. ^ *Jonathan Corbet (April 27, 2004). "Being honest with MODULE_LICENSE". LWN.net. Archived from the original on November 2, 2012. Retrieved October 30, 2012.*
11. ^ Exploiting Loadable Kernel Modules Archived 2012-02-04 at the Wayback Machine
12. ^ *"Sysctl/kernel.txt". Retrieved January 4, 2013.* `{{cite web}}` : CS1 maint: deprecated archival service (link)
13. ^ *Kees Cook (2012-11-28). "Clean module disabling". outflux.net. Retrieved 2020-10-05.*
14. ^ *"Kernel Extensions". Mac Developer Library. Apple. September 16, 2015. Archived from the original on August 17, 2016. Retrieved September 29, 2016.*
15. ^ *"Info.plist Properties for Kernel Extensions". Apple Inc. Archived from the original on September 26, 2012. Retrieved September 27, 2012.*
16. ^ `kextload(8)` – Darwin and macOS System Manager's Manual
