Features

Matrix

By Default
Available
Unimplemented

feature	20.04 LTS	22.04 LTS	24.04 LTS	24.10
No Open Ports	policy	policy	policy	policy
Password hashing	sha512	yescrypt	yescrypt	yescrypt
SYN cookies	kernel & sysctl	kernel & sysctl	kernel & sysctl	kernel & sysctl
Automatic security updates	enabled	enabled	enabled	enabled
Kernel Livepatches	20.04 LTS Kernel	22.04 LTS Kernel	--	--
Disable legacy TLS	policy	policy	policy	policy
Filesystem Capabilities	kernel & userspace (default on server)	kernel & userspace (default on server)	kernel & userspace (default on server)	kernel & userspace (default on server)
Configurable Firewall	ufw	ufw	ufw	ufw

Cloud PRNG seed	pollinate	pollinate	pollinate	pollinate
PR_SET_SECCOMP	kernel	kernel	kernel	kernel
AppArmor	2.13.3	3.0.4	3.0.7	3.0.7
AppArmor unprivileged user namespace restrictions	--	--	kernel & userspace	kernel & userspace
SELinux	universe	universe	universe	universe
SMACK	kernel	kernel	kernel	kernel
Encrypted LVM	main installer	main installer	main installer	main installer
File Encryption	ZFS dataset encryption available, encrypted Home (eCryptfs) and ext4 encryption (fscrypt) available in universe	ZFS dataset encryption available, encrypted Home (eCryptfs) and ext4 encryption (fscrypt) available in universe	ZFS dataset encryption available, encrypted Home (eCryptfs) and ext4 encryption (fscrypt) available in universe	ZFS dataset encryption available, encrypted Home (eCryptfs) and ext4 encryption (fscrypt) available in universe
Trusted Platform Module	kernel & userspace (tpm-tools)	kernel & userspace (tpm-tools)	kernel & userspace (tpm-tools)	kernel & userspace (tpm-tools)
Stack Protector	gcc patch	gcc patch	gcc patch	gcc patch
Heap Protector	glibc	glibc	glibc	glibc
Pointer Obfuscation	glibc	glibc	glibc	glibc
Stack ASLR	kernel	kernel	kernel	kernel
Libs/mmap ASLR	kernel	kernel	kernel	kernel
Exec ASLR	kernel	kernel	kernel	kernel
brk ASLR	kernel	kernel	kernel	kernel
VDSO ASLR	kernel	kernel	kernel	kernel

Built as PIE	gcc patch (amd64, ppc64el, s390x), package list for others	gcc patch (amd64, ppc64el, s390x), package list for others	gcc patch (amd64, ppc64el, s390x), package list for others	gcc patch (amd64, ppc64el, s390x), package list for others
Built with Fortify Source	gcc patch	gcc patch	gcc patch	gcc patch
Built with RELRO	gcc patch	gcc patch	gcc patch	gcc patch
Built with BIND_NOW	gcc patch (amd64, ppc64el, s390x), package list for others	gcc patch (amd64, ppc64el, s390x), package list for others	gcc patch (amd64, ppc64el, s390x), package list for others	gcc patch (amd64, ppc64el, s390x), package list for others
Built with -fstack-clash-protection	gcc patch (i386, amd64, ppc64el, s390x)	gcc patch (i386, amd64, ppc64el, s390x)	gcc patch (i386, amd64, ppc64el, s390x)	gcc patch (i386, amd64, ppc64el, s390x)
Built with -fcf-protection	gcc patch (i386, amd64)	gcc patch (i386, amd64)	gcc patch (i386, amd64)	gcc patch (i386, amd64)
Non-Executable Memory	PAE, ia32 partial-NX-emulation	PAE, ia32 partial-NX-emulation	PAE, ia32 partial-NX-emulation	PAE, ia32 partial-NX-emulation
/proc/$pid/maps protection	kernel	kernel	kernel	kernel
Symlink restrictions	kernel	kernel	kernel	kernel
Hardlink restrictions	kernel	kernel	kernel	kernel
FIFO restrictions	kernel & sysctl	kernel & sysctl	kernel & sysctl	kernel & sysctl
Regular file restrictions	kernel & sysctl	kernel & sysctl	kernel & sysctl	kernel & sysctl
ptrace scope	kernel	kernel	kernel	kernel
0-address protection	kernel	kernel	kernel	kernel
/dev/mem protection	kernel	kernel	kernel	kernel
/dev/kmem disabled	kernel	kernel	kernel	kernel

Block module loading	sysctl	sysctl	sysctl	sysctl
Read-only data sections	kernel	kernel	kernel	kernel
Stack protector	kernel	kernel	kernel	kernel
Module RO/NX	kernel	kernel	kernel	kernel
Kernel Address Display Restriction	kernel	kernel	kernel	kernel
Kernel Address Space Layout Randomisation	kernel (i386, amd64, arm64, and s390 only)	kernel (i386, amd64, arm64, and s390 only)	kernel (i386, amd64, arm64, and s390 only)	kernel (i386, amd64, arm64, and s390 only)
Denylist Rare Protocols	kernel	kernel	kernel	kernel
Syscall Filtering	kernel	kernel	kernel	kernel
dmesg restrictions	sysctl	kernel	kernel	kernel
Block kexec	sysctl	sysctl	sysctl	sysctl
UEFI Secure Boot (amd64)	amd64, kernel signature enforcement	amd64, kernel signature enforcement	amd64, kernel signature enforcement	amd64, kernel signature enforcement
usbguard	kernel & userspace	kernel & userspace	kernel & userspace	kernel & userspace
usbauth	kernel & userspace	kernel & userspace	kernel & userspace	kernel & userspace
bolt	kernel & userspace	kernel & userspace	kernel & userspace	kernel & userspace
thunderbolt-tools	kernel & userspace	kernel & userspace	kernel & userspace	kernel & userspace
Kernel Lockdown	integrity only, no confidentiality	integrity only, no confidentiality	integrity only, no confidentiality	integrity only, no confidentiality

Features

Configuration

Contents

No Open Ports

Default installations of Ubuntu must have no listening network services after initial install. Exceptions to this rule on desktop systems include network infrastructure services such as a DHCP client and mDNS (Avahi/ZeroConf, see ZeroConfPolicySpec for implementation details and justification). For Ubuntu in the cloud, exceptions include network infrastructure services for the cloud and OpenSSH running with client public key and port access configured by the cloud provider. When installing Ubuntu Server, the administrator can, of course, select specific services to install beyond the defaults (e.g. Apache).

Testing for this can be done with `netstat -an --inet | grep LISTEN | grep -v 127.0.0.1:` on a fresh install.

Password hashing

The system password used for logging into Ubuntu is stored in /etc/shadow. Very old style password hashes were based on DES and visible in /etc/passwd. Modern Linux has long since moved to /etc/shadow, and for some time now has used salted MD5-based hashes for password verification (crypt id 1). Since MD5 is considered "broken" for some uses and as computational power available to perform brute-forcing of MD5 increases, Ubuntu 8.10 and later proactively moved to using salted SHA-512 based password hashes (crypt id 6), which are orders of magnitude more difficult to brute-force. Ubuntu 22.04 LTS and later then moved to yescrypt to provide increased protection against offline password cracking. See the Manpage:crypt manpage for additional details.

See test-glibc-security.py for regression tests.

SYN cookies

When a system is overwhelmed by new network connections, SYN cookie use is activated, which helps mitigate a SYN-flood attack.

See test-kernel-security.py for configuration regression tests.

Automatic security updates

Starting with Ubuntu 16.04 LTS, unattended-upgrades is configured to automatically apply security updates daily. Earlier Ubuntu releases can be configured to automatically apply security updates.

Kernel Livepatches

The Canonical Livepatch service provides security fixes for most major kernel security issues without requiring a reboot. Ubuntu users can take advantage of the service on up to three nodes for free. All machines covered by an Ubuntu Advantage support subscription are able to receive livepatches.

Disable legacy TLS

Legacy versions of the Transport Layer Security protocol including SSL 3.0, TLS 1.0 and TLS 1.1, have several inherent vulnerabilities and cannot provide the advertised level of security. For that Ubuntu 20.04 and later proactively disable these versions setting the bar of secure communication to protocols that are considered secure today.

To communicate with legacy systems it is possible to re-enable the protocols. See this discourse article for more information.

Subsystems

Filesystem Capabilities

The need for setuid applications can be reduced via the application of filesystem capabilities using the xattrs available to most modern filesystems. This reduces the possible misuse of vulnerable setuid applications. The kernel provides the support, and the user-space tools are in main ("libcap2-bin").

See test-kernel-security.py for configuration regression tests.

Configurable Firewall

ufw is a frontend for iptables, and is installed by default in Ubuntu (users must explicitly enable it). Particularly well-suited for host-based firewalls, ufw provides a framework for managing a netfilter firewall, as well as a command-line interface for manipulating the firewall. ufw aims to provide an easy to use interface for people unfamiliar with firewall concepts, while at the same time simplifies complicated iptables commands to help an administrator who knows what he or she is doing. ufw is an upstream for other distributions and graphical frontends.

See ufw tests for regression tests.

Cloud PRNG seed

Pollinate is a client application that retrieves entropy from one or more Pollen servers and seeds the local Pseudo Random Number Generator (PRNG). Pollinate is designed to adequately and securely seed the PRNG through communications with a Pollen server which is particularly important for systems operating in cloud environments. Starting with Ubuntu 14.04 LTS, Ubuntu cloud images include the Pollinate client, which will try to seed the PRNG with input from https://entropy.ubuntu.com for up to 3 seconds on first boot.

See pollen_test.go for regression tests

PR_SET_SECCOMP

Setting SECCOMP for a process is meant to confine it to a small subsystem of system calls, used for specialized processing-only programs.

See test-kernel-security.py for regression tests.

Mandatory Access Control (MAC)

Mandatory Access Controls are handled via the kernel LSM hooks.

AppArmor

AppArmor is a path-based MAC. It can mediate:

- file access (read, write, link, lock)
- library loading
- execution of applications
- coarse-grained network (protocol, type, domain)
- capabilities
- coarse owner checks (task must have the same euid/fsuid as the object being checked) starting with Ubuntu 9.10
- mount starting with Ubuntu 12.04 LTS
- unix(7) named sockets starting with Ubuntu 13.10
- DBus API (path, interface, method) starting with Ubuntu 13.10
- signal(7) starting with Ubuntu 14.04 LTS
- ptrace(2) starting with Ubuntu 14.04 LTS
- unix(7) abstract and anonymous sockets starting with Ubuntu 14.10

AppArmor is a core technology for application confinement for Ubuntu Touch and Snappy for Ubuntu Core and Personal.

Example profiles are found in the apparmor-profiles package from universe, and by-default shipped enforcing profiles are being built up:

Source package/binary	**12.04 LTS**	**14.04 LTS**	**16.04 LTS**	**18.04 LTS**	**20.04 LTS**	**20.10**
Akonadi (mysqld)	yes	yes	yes	yes	yes	yes
Apache (apache2)	yes[1]	yes[1]	yes[1]	yes[1]	yes	yes
Bind (named)	yes	yes	yes	yes	yes	yes
ClamAV (clamd,freshclam)	yes	yes	yes	yes	yes	yes
Cups (cupsd)	yes	yes	yes	yes	yes	yes
Evince	yes	yes	yes	yes	yes	yes
Firefox (firefox-3.5/firefox)	yes[1]	yes[1]	yes[1]	yes[1]	yes	yes
gdm-guest-session	N/A	N/A	yes	yes	yes	yes
ISC Dhcpd (dhcpd3/dhcpd)	yes	yes	yes	yes	yes	yes
ISC Dhcp client (dhclient3/dhclient)	yes	yes	yes	yes	yes	yes
juju	yes[2]	yes[2]	yes[2]	yes[2]	yes	yes
Libvirt (libvirtd and kvm/qemu guests)	yes	yes	yes	yes	yes	yes

Lightdm guest session	yes	yes	yes	--	--	--
LXC	yes[3]	yes[3]	yes[3]	yes[3]	yes	yes
MAAS dhcpd (dhcpd)	yes	yes	yes	yes	--	--
MySQL (mysqld)	yes	yes	yes	yes	yes	yes
NTP (ntpd)	yes	yes	yes	--	--	--
OpenLDAP (slapd)	yes	yes	yes	yes	yes	yes
quassel-core	yes	yes	yes	yes	yes	yes
rsyslog	yes[1]	yes[1]	yes[1]	yes[1]	yes	yes
tcpdump	yes	yes	yes	yes	yes	yes
Telepathy	yes	yes	yes	--	--	--
AppStore apps (click)[4]	--	yes	yes	--	--	--
Cups filters (cups-browsed)	--	yes	yes	yes	yes	yes
lightdm-remote-session-freerdp	--	yes	yes	--	--	--
lightdm-remote-session-uccsconfigure	--	yes	yes	--	--	--
media-hub	--	yes	yes	--	--	--
mediascanner2	--	yes	yes	--	--	--
squid3	--	yes[1]	yes[1]	yes[1]	yes	yes
sssd	--	yes[1]	yes[1]	yes[1]	yes	yes
StrongSwan (stroke/lookip)	--	yes	yes	yes	yes	yes
Telepathy (ofono)	--	yes	yes	yes	yes	yes

AppStore apps (snappy)[5]	--	--	yes	yes	yes	yes
libvirt (libvirt-lxc containers)	--	--	yes	yes	yes	yes
LXD	--	--	yes	yes	yes	yes
snap-confine (aka ubuntu-core-launcher)	--	--	yes	yes	yes	yes
ubuntu-download-manager (extractor)	--	--	yes	--	--	--
webbrowser-app	--	--	yes	--	--	--
chrony	--	--	--	yes	yes	yes
ippusbxd	--	--	--	yes	yes	yes
libreoffice[6]	--	--	--	yes	yes	yes
man-db	--	--	--	yes	yes	yes
mozc	--	--	--	yes	yes	yes
anope	--	--	--	--	--	yes

1. Disabled by default and be opt-in for advanced users
2. https://juju.ubuntu.com/AppArmor
3. Preliminary support
4. Ubuntu Touch apps in the Ubuntu AppStore are confined with AppArmor by default. See ApplicationConfinement for details
5. Apps in the Ubuntu AppStore are confined with AppArmor by default. See the security guide for details
6. Mixture of enforce and complain mode profiles

Starting with Ubuntu 16.10, AppArmor can "stack" profiles so that the mediation decisions are made using the intersection of multiple profiles. This feature, combined with AppArmor profile namespaces, allows LXD to define a profile that an entire container will be confined with while still allowing individual, containerized processes to be further confined with profiles loaded inside of the container environment.

See test-apparmor.py and test-kernel-security.py for regression tests.

AppArmor unprivileged user namespace restrictions

Starting with Ubuntu 23.10, AppArmor provides support for denying unprivileged applications the use of user namespaces. This prevents an unprivileged application from making use of a user namespace to gain access to additional capabilities and various kernel subsystems which present an additional attack surface. Applications which do require legitimate unprivileged access to user namespaces are designated by an appropriate AppArmor profile. Starting with Ubuntu 24.04 this is enabled by default.

See test-apparmor.py for regression tests.

SELinux

SELinux is an inode-based MAC. Targeted policies are available for Ubuntu in universe. Installing the "selinux" package will make the boot-time adjustments that are needed.

See test-kernel-security.py for configuration regression tests.

SMACK

SMACK is a flexible inode-based MAC.

See test-kernel-security.py for configuration regression tests.

Storage Encryption

Encrypted LVM

Ubuntu 12.10 and newer include the ability to install Ubuntu onto an encrypted LVM, which allows all partitions in the logical volume, including swap, to be encrypted. Between 6.06 LTS and 12.04 LTS the alternate installer can install to an encrypted LVM.

File Encryption

Encrypted Private Directories were implemented, utilizing eCryptfs, in Ubuntu 8.10 as a secure location for users to store sensitive information. The server and alternate installers had the option to setup an encrypted private directory for the first user. In Ubuntu 9.04, support for encrypted home and filename encryption was added. Encrypted Home allowed users to encrypt all files in their home directory and was supported in the Alternate Installer and also in the Desktop Installer via the preseed option `user-setup/encrypt-home=true`.

Official support for Encrypted Private and Encrypted Home directories was dropped in Ubuntu 18.04 LTS. It is still possible to configure an encrypted private or home directory, after Ubuntu is installed, with the `ecryptfs-setup-private` utility provided by the `ecryptfs-utils` package.

Starting in Ubuntu 18.04 LTS, it is also possible to install and use fscrypt to encrypt directories on ext4 filesystems. Note that fscrypt is not officially supported but is available via the fscrypt package in universe.

Trusted Platform Module

TPM 1.2 support was added in Ubuntu 7.10. "tpm-tools" and related libraries are available in Ubuntu universe. For TPM 2.0, tpm2-tools is available in Ubuntu universe.

Userspace Hardening

Many security features are available through the default compiler flags used to build packages and through the kernel in Ubuntu. **Note:** Ubuntu's compiler hardening applies not only to its official builds but also anything built on Ubuntu using its compiler.

Stack Protector

gcc's -fstack-protector provides a randomized stack canary that protects against stack overflows, and reduces the chances of arbitrary code execution via controlling return address destinations. Enabled at compile-time. (A small number of applications do not play well with it, and have it disabled.) The routines used for stack checking are actually part of glibc, but gcc is patched to enable linking against those routines by default.

See test-gcc-security.py for regression tests.

Heap Protector

The GNU C Library heap protector (both automatic via ptmalloc and manual) provides corrupted-list/unlink/double-free/overflow protections to the glibc heap memory manager (first introduced in glibc 2.3.4). This stops the ability to perform arbitrary code execution via heap memory overflows that try to corrupt the control structures of the malloc heap memory areas.

This protection has evolved over time, adding more and more protections as additional corner-cases were researched. As it currently stands, glibc 2.10 and later appears to successfully resist even these hard-to-hit conditions.

See test-glibc-security.py for regression tests.

Pointer Obfuscation

Some pointers stored in glibc are obfuscated via PTR_MANGLE/PTR_UNMANGLE macros internally in glibc, preventing libc function pointers from being overwritten during runtime.

See test-glibc-security.py for regression tests.

Address Space Layout Randomisation (ASLR)

ASLR is implemented by the kernel and the ELF loader by randomising the location of memory allocations (stack, heap, shared libraries, etc). This makes memory addresses harder to predict when an attacker is attempting a memory-corruption exploit. ASLR is controlled system-wide by the value of `/proc/sys/kernel/randomize_va_space`. Prior to Ubuntu 8.10, this defaulted to "1" (on). In later releases that included brk ASLR, it defaults to "2" (on, with brk ASLR).

See test-kernel-security.py for regression tests for all the different types of ASLR.

Stack ASLR

Each execution of a program results in a different stack memory space layout. This makes it harder to locate in memory where to attack or deliver an executable attack payload. This was available in the mainline kernel since 2.6.15 (Ubuntu 6.06).

Libs/mmap ASLR

Each execution of a program results in a different mmap memory space layout (which causes the dynamically loaded libraries to get loaded into different locations each time). This makes it harder to locate in memory where to jump to for "return to libc" to similar attacks. This was available in the mainline kernel since 2.6.15 (Ubuntu 6.06).

Exec ASLR

Each execution of a program that has been built with "-fPIE -pie" will get loaded into a different memory location. This makes it harder to locate in memory where to attack or jump to when performing memory-corruption-based attacks. This was available in the mainline kernel since 2.6.25 (and was backported to Ubuntu 8.04 LTS).

brk ASLR

Similar to exec ASLR, brk ASLR adjusts the memory locations relative between the exec memory area and the brk memory area (for small mallocs). The randomization of brk offset from exec memory was added in 2.6.26 (Ubuntu 8.10), though some of the effects of brk ASLR can be seen for PIE programs in Ubuntu 8.04 LTS since exec was ASLR, and brk is allocated immediately after the exec region (so it was technically randomized, but not randomized with respect to the text region until 8.10).

VDSO ASLR

Each execution of a program results in a random vdso location. While this has existed in the mainline kernel since 2.6.18 (x86, PPC) and 2.6.22 (x86_64), it hadn't been enabled in Ubuntu 6.10 due to COMPAT_VDSO being enabled, which was removed in Ubuntu 8.04 LTS. This protects against jump-into-syscall attacks. Only x86 (maybe ppc?) is supported by glibc 2.6. glibc 2.7 (Ubuntu 8.04 LTS) supports x86_64 ASLR vdso. People needing ancient pre-libc6 static high vdso mappings can use "vdso=2" on the kernel boot command line to gain COMPAT_VDSO again.

- https://lwn.net/Articles/184734/
- https://articles.manugarg.com/systemcallinlinux2_6.html

Built as PIE

All programs built as Position Independent Executables (PIE) with "-fPIE -pie" can take advantage of the exec ASLR. This protects against "return-to-text" and generally frustrates memory corruption attacks. This requires centralized changes to the compiler options when building the entire archive. PIE has a large (5-10%) performance penalty on architectures with small numbers of general registers (e.g. x86), so it initially was only used for a select number of security-critical packages (some upstreams natively support building with PIE, other require the use of "hardening-wrapper" to force on the correct compiler and linker flags). PIE on 64-bit architectures do not have the same penalties, and it was made the default (as of 16.10, it is the default on amd64, ppc64el and s390x). As of 17.10, it was decided that the security benefits are significant enough that PIE is now enabled across all architectures in the Ubuntu archive by default.

See test-built-binaries.py for regression tests.

Built with Fortify Source

Programs built with "-D_FORTIFY_SOURCE=2" (and -O1 or higher), enable several compile-time and run-time protections in glibc:

- expand unbounded calls to "sprintf", "strcpy" into their "n" length-limited cousins when the size of a destination buffer is known (protects against memory overflows).
- stop format string "%n" attacks when the format string is in a writable memory segment.
- require checking various important function return codes and arguments (e.g. system, write, open).
- require explicit file mask when creating new files.

See test-gcc-security.py for regression tests.

Built with RELRO

Hardens ELF programs against loader memory area overwrites by having the loader mark any areas of the relocation table as read-only for any symbols resolved at load-time ("read-only relocations"). This reduces the area of possible GOT-overwrite-style memory corruption attacks.

See test-gcc-security.py for regression tests.

Built with BIND_NOW

Marks ELF programs to resolve all dynamic symbols at start-up (instead of on-demand, also known as "immediate binding") so that the GOT can be made entirely read-only (when combined with RELRO above).

See test-built-binaries.py for regression tests.

Built with -fstack-clash-protection

Adds extra instructions around variable length stack memory allocations (via alloca() or gcc variable length arrays etc) to probe each page of memory at allocation time. This mitigates stack-clash attacks by ensuring all stack memory allocations are valid (or by raising a segmentation fault if they are not, and turning a possible code-execution attack into a denial of service).

See test-built-binaries.py for regression tests.

Built with -fcf-protection

Instructs the compiler to generate instructions to support Intel's Control-flow Enforcement Technology (CET).

See test-built-binaries.py for regression tests.

Non-Executable Memory

Most modern CPUs protect against executing non-executable memory regions (heap, stack, etc). This is known either as Non-eXecute (NX) or eXecute-Disable (XD), and some BIOS manufacturers needlessly disable it by default, so check your BIOS Settings. This protection reduces the areas an attacker can use to perform arbitrary code execution. It requires that the kernel use "PAE" addressing (which also allows addressing of physical addresses above 3GB). The 64bit and 32bit `-server` and `-generic-pae` kernels are compiled with PAE addressing. Starting in Ubuntu 9.10, this protection is partially emulated for processors lacking NX when running on a 32bit kernel (built with or without PAE). After booting, you can see what NX protection is in effect:

- Hardware-based (via PAE mode):

  ```
  [    0.000000] NX (Execute Disable) protection: active
  ```

- Partial Emulation (via segment limits):

  ```
  [    0.000000] Using x86 segment limits to approximate NX protection
  ```

If neither are seen, you do not have any NX protections enabled. Check your BIOS settings and CPU capabilities. If "nx" shows up in each of the "flags" lines in `/proc/cpuinfo`, it is enabled/supported by your hardware (and a PAE kernel is needed to actually use it).

Starting in Ubuntu 11.04, BIOS NX settings are ignored by the kernel.

Ubuntu 9.04 and earlier

		CPU supports NX		CPU lacks NX
		BIOS enables NX	BIOS disables NX	
i386	`-386`, `-generic` kernel (non-PAE)	nx unsupported	nx unsupported	nx unsupported
	`-server` kernel (PAE)	real nx	nx unsupported	nx unsupported
amd64 any kernel (PAE)		real nx	nx unsupported	N/A

Ubuntu 9.10 through 10.10

		CPU supports NX		CPU lacks NX
		BIOS enables NX	BIOS disables NX	
i386	`-386`, `-generic` kernel (non-PAE)	nx-emulation	nx-emulation	nx-emulation
	`-server`, `-generic-pae` kernel (PAE)	real nx	nx-emulation	nx-emulation
amd64 any kernel (PAE)		real nx	nx unsupported	N/A

Ubuntu 11.04 and later

		CPU supports NX	CPU lacks NX
i386	`-386`, `-generic` kernel (non-PAE)	nx-emulation	nx-emulation
	`-server`, `-generic-pae` kernel (PAE)	real nx	nx-emulation
amd64 any kernel (PAE)		real nx	N/A

See test-kernel-security.py for regression tests.

/proc/$pid/maps protection

With ASLR, a process's memory space layout suddenly becomes valuable to attackers. The "maps" file is made read-only except to the process itself or the owner of the process. Went into mainline kernel with sysctl toggle in 2.6.22. The toggle was made non-optional in 2.6.27, forcing the privacy to be enabled regardless of sysctl settings (this is a good thing).

See test-kernel-security.py for regression tests.

Symlink restrictions

A long-standing class of security issues is the symlink-based ToCToU race, most commonly seen in world-writable directories like `/tmp/`. The common method of exploitation of this flaw is crossing privilege boundaries when following a given symlink (i.e. a `root` user follows a symlink belonging to another user).

In Ubuntu 10.10 and later, symlinks in world-writable sticky directories (e.g. `/tmp`) cannot be followed if the follower and directory owner do not match the symlink owner. The behavior is controllable through the `/proc/sys/kernel/yama/protected_sticky_symlinks` sysctl, available via Yama.

See test-kernel-security.py for regression tests.

Hardlink restrictions

Hardlinks can be abused in a similar fashion to symlinks above, but they are not limited to world-writable directories. If `/etc/` and `/home/` are on the same partition, a regular user can create a hardlink to `/etc/shadow` in their home directory. While it retains the original owner and permissions, it is possible for privileged programs that are otherwise symlink-safe to mistakenly access the file through its hardlink. Additionally, a very minor untraceable quota-bypassing local denial of service is possible by an attacker exhausting disk space by filling a world-writable directory with hardlinks.

In Ubuntu 10.10 and later, hardlinks cannot be created to files that the user would be unable to read and write originally, or are otherwise sensitive. The behavior is controllable through the `/proc/sys/kernel/yama/protected_nonaccess_hardlinks` sysctl, available via Yama.

See test-kernel-security.py for regression tests.

FIFO restrictions

Processes may not check that the files being created are actually created as the desired type. This global control forbids some potentially unsafe configurations from working.

See the kernel admin-guide for documentation.

Regular file restrictions

Processes may not check that the files being created are actually created as desired. This global control forbids some potentially unsafe configurations from working.

See the kernel admin-guide for documentation.

ptrace scope

A troubling weakness of the Linux process interfaces is that a single user is able to examine the memory and running state of any of their processes. For example, if one application was compromised, it would be possible for an attacker to attach to other running processes (e.g. SSH sessions, GPG agent, etc) to extract additional credentials and continue to immediately expand the scope of their attack without resorting to user-assisted phishing or trojans.

In Ubuntu 10.10 and later, users cannot ptrace processes that are not a descendant of the debugger. The behavior is controllable through the `/proc/sys/kernel/yama/ptrace_scope` sysctl, available via Yama.

In the case of automatic crash handlers, a crashing process can specficially allow an existing crash handler process to attach on a process-by-process basis using `prctl(PR_SET_PTRACER, debugger_pid, 0, 0, 0)`.

See test-kernel-security.py for regression tests.

Kernel Hardening

The kernel itself has protections enabled to make it more difficult to become compromised.

0-address protection

Since the kernel and userspace share virtual memory addresses, the "NULL" memory space needs to be protected so that userspace mmap'd memory cannot start at address 0, stopping "NULL dereference" kernel attacks. This is possible with 2.6.22 kernels, and was implemented with the "mmap_min_addr" sysctl setting. Since Ubuntu 9.04, the mmap_min_addr setting is built into the kernel. (64k for x86, 32k for ARM.)

See test-kernel-security.py for regression tests.

/dev/mem protection

Some applications (Xorg) need direct access to the physical memory from user-space. The special file `/dev/mem` exists to provide this access. In the past, it was possible to view and change kernel memory from this file if an attacker had root access. The CONFIG_STRICT_DEVMEM kernel option was introduced to block non-device memory access (originally named CONFIG_NONPROMISC_DEVMEM).

See test-kernel-security.py for regression tests.

/dev/kmem disabled

There is no modern user of `/dev/kmem` any more beyond attackers using it to load kernel rootkits. CONFIG_DEVKMEM is set to "n". While the `/dev/kmem` device node still exists in Ubuntu 8.04 LTS through Ubuntu 9.04, it is not actually attached to anything in the kernel.

See test-kernel-security.py for regression tests.

Block module loading

In Ubuntu 8.04 LTS and earlier, it was possible to remove CAP_SYS_MODULES from the system-wide capability bounding set, which would stop any new kernel modules from being loaded. This was another layer of protection to stop kernel rootkits from being installed. The 2.6.25 Linux kernel (Ubuntu 8.10) changed how bounding sets worked, and this functionality disappeared. Starting with Ubuntu 9.10, it is now possible to block module loading again by setting "1" in `/proc/sys/kernel/modules_disabled`.

See test-kernel-security.py for regression tests.

Read-only data sections

This makes sure that certain kernel data sections are marked to block modification. This helps protect against some classes of kernel rootkits. Enabled via the CONFIG_DEBUG_RODATA option.

See test-kernel-security.py for configuration regression tests.

Stack protector

Similar to the stack protector used for ELF programs in userspace, the kernel can protect its internal stacks as well. Enabled via the CONFIG_CC_STACKPROTECTOR option.

See test-kernel-security.py for configuration regression tests.

Module RO/NX

This feature extends CONFIG_DEBUG_RODATA to include similar restrictions for loaded modules in the kernel. This can help resist future kernel exploits that depend on various memory regions in loaded modules. Enabled via the CONFIG_DEBUG_MODULE_RONX option.

See test-kernel-security.py for configuration regression tests.

Kernel Address Display Restriction

When attackers try to develop "run anywhere" exploits for kernel vulnerabilities, they frequently need to know the location of internal kernel structures. By treating kernel addresses as sensitive information, those locations are not visible to regular local users. Starting with Ubuntu 11.04, `/proc/sys/kernel/kptr_restrict` is set to "1" to block the reporting of known kernel address leaks. Additionally, various files and directories were made readable only by the root user: `/boot/vmlinuz*`, `/boot/System.map*`, `/sys/kernel/debug/`, `/proc/slabinfo`

See test-kernel-security.py for regression tests.

Kernel Address Space Layout Randomisation

Kernel Address Space Layout Randomisation (kASLR) aims to make some kernel exploits more difficult to implement by randomizing the base address value of the kernel. Exploits that rely on the locations of internal kernel symbols must discover the randomized base address.

kASLR is available starting with Ubuntu 14.10 and is enabled by default in 16.10 and later.

Before 16.10, you can specify the "kaslr" option on the kernel command line to use kASLR.

Note: Before 16.10, enabling kASLR will disable the ability to enter hibernation mode.

Denylist Rare Protocols

Normally the kernel allows all network protocols to be autoloaded on demand via the `MODULE_ALIAS_NETPROTO(PF_...)` macros. Since many of these protocols are old, rare, or generally of little use to the average Ubuntu user and may contain undiscovered exploitable vulnerabilities, they have been denylisted since Ubuntu 11.04. These include: ax25, netrom, x25, rose, decnet, econet, rds, and af_802154. If any of the protocols are needed, they can speficially loaded via modprobe, or the `/etc/modprobe.d/blacklist-rare-network.conf` file can be updated to remove the denylist entry.

See test-kernel-security.py for regression tests.

Syscall Filtering

Programs can filter out the availability of kernel syscalls by using the seccomp_filter interface. This is done in containers or sandboxes that want to further limit the exposure to kernel interfaces when potentially running untrusted software.

See test-kernel-security.py for regression tests.

dmesg restrictions

When attackers try to develop "run anywhere" exploits for vulnerabilties, they frequently will use dmesg output. By treating dmesg output as sensitive information, this output is not available to the attacker. Starting with Ubuntu 12.04 LTS, `/proc/sys/kernel/dmesg_restrict` can be set to "1" to treat dmesg output as sensitive. Starting with 20.10, this is enabled by default.

Block kexec

Starting with Ubuntu 14.04 LTS, it is now possible to disable kexec via sysctl. CONFIG_KEXEC is enabled in Ubuntu so end users are able to use kexec as desired and the new sysctl allows administrators to disable kexec_load. This is desired in environments where CONFIG_STRICT_DEVMEM and modules_disabled are set, for example. When Secure Boot is in use, kexec is restricted by default to only load appropriately signed and trusted kernels.

UEFI Secure Boot (amd64)

Starting with Ubuntu 12.04 LTS, UEFI Secure Boot was implemented in enforcing mode for the bootloader and non-enforcing mode for the kernel. With this configuration, a kernel that fails to verify will boot without UEFI quirks enabled. The Ubuntu 18.04.2 release of Ubuntu 18.04 LTS enabled enforcing mode for the bootloader and the kernel, so that kernels which fail to verify will not be booted, and kernel modules which fail to verify will not be loaded. This is planned to be backported for Ubuntu 16.04 LTS and Ubuntu 14.04 LTS (however only with kernel signature enforcement for Ubuntu 14.04 LTS, not kernel module signature enforcement).

usbguard

Starting with Ubuntu 16.10, the usbguard package has been available in universe to provide a tool for using the Linux kernel's USB authorization support, to control device IDs and device classes that will be recognized.

usbauth

Starting with Ubuntu 18.04, the usbauth package has been available in universe to provide a tool for using the Linux kernel's USB authorization support, to control device IDs and device classes that will be recognized.

bolt

Starting with Ubuntu 18.04, the bolt package has been available in main to provide a desktop-oriented tool for using the Linux kernel's Thunderbolt authorization support.

thunderbolt-tools

Starting with Ubuntu 18.04, the thunderbolt-tools package has been available in universe to provide a server-oriented tool for using the Linux kernel's Thunderbolt authorization support.

Kernel Lockdown

Starting with Ubuntu 20.04, the Linux kernel's lockdown mode is enabled in integrity mode. This prevents the root account from loading arbitrary modules or BPF programs that can manipulate kernel datastructures. Lockdown enforcement is tied to UEFI secure boot.

Additional Documentation

- Coordination with Debian: https://wiki.debian.org/Hardening
- Gentoo's Hardening project: https://www.gentoo.org/proj/en/hardened/hardened-toolchain.xml
- Ubuntu Security Features for all releases

If you have questions or comments on these features, please contact the security team.

CategorySecurityTeam

Security/Features (last edited 2024-07-12 08:27:03 by 0xnishit @ undefined.hostname.localhost[2402:a00:184:6711:2285:a917:746c:c49f]:0xnishit)