Ubuntu 14.04 LTS
Security for Human Beings

Oak Ridge National Labs
Cyber and Information
Security Research Conference
April 9, 2014
We are the company behind Ubuntu
What we offer

- **Ubuntu**
  The Cloud OS

- **OpenStack**
  The Cloud Platform

- **Cloud Orchestration**
  Juju, Maas, Landscape

- **Professional Services**

- **World-Class Enterprise Support**
Dustin Kirkland

- Ubuntu Cloud Product Manager at Canonical
- Ubuntu Core Developer
- Author and Maintainer of a few dozen open source packages
  - byobu, encryptfs-utils, ssh-import-id, update-motd, dotdee, bikeshed, bootmail, testdrive, powernap, run-one, pollen, pollinate, ...
- Linux security professional with 15 years experience
- Father, husband, dog lover, homebrewer

Twitter: @dustinkirkland
Blog: http://blog.dustinkirkland.com
LinkedIn: http://www.linkedin.com/in/dustinkirkland
GPG: 4096R/E2D9 E1C5 F9F5 D592 91F4 607D 95E6 4373 F152 9469

https://wiki.ubuntu.com/Security/Features
The key takeaway is that we have thought about all of this for you... so that you don’t have to!
No Open Ports

Default installations of Ubuntu must have no listening network services after initial install. Exceptions to this rule include network infrastructure services such as the DHCP client and mDNS (Avahi/ZeroConf, see ZeroConfPolicySpec for implementation details and justification). 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.

https://wiki.ubuntu.com/Security/Features
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. See the 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 mitigates SYN-flood DoS attacks.

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

https://wiki.ubuntu.com/Security/Features
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.

https://wiki.ubuntu.com/Security/Features
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 (for up to 3 seconds at first boot) to seed the PRNG with input from https://entropy.ubuntu.com.

See pollen_test.go for regression tests

https://wiki.ubuntu.com/Security/Features
PR_SET_SECCOMP

Setting SECCOMP for a process confines a process to a small subsystem of system calls, used for specialized processing-only programs. OpenSSH Server and VSFTPD are two examples of programs that use this feature.

See test-kernel-security.py for regression tests.
**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
- mount
- named sockets
- DBus API (path, interface, method) starting with Ubuntu 13.10
- signal(7), ptrace(2)
- coarse owner checks (task must have the same euid/fsuid as the object being checked)

Example profiles are found in the apparmor-profiles package from universe, and **Ubuntu ships, by-default, a 20+ enforcing profiles**

- CUPS, OpenLDAP, MySQL, Bind, ClamAV, tcpdump, DHCP, Evince, NTP, Firefox, Libvirt, Apache2, Telepathy, Lightdm, juju, rsyslog, LCX, squid3, Click apps, Pollen, etc.

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

https://wiki.ubuntu.com/Security/Features
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.

https://wiki.ubuntu.com/Security/Features
**eCryptFS**

Encrypted Private Directories were implemented 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. As of Ubuntu 9.04, support for encrypted home was added, allowing users to encrypt all files in their home directory. Encrypted Home is supported in the Alternate Installer, and available in the Desktop Installer via the preseed option user-setup/encrypt-home=true. Also, the Ubuntu 9.04 kernel carries a patchset for eCryptfs to support encrypted filenames.

https://wiki.ubuntu.com/Security/Features
Userspace Toolchain 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.

https://wiki.ubuntu.com/Security/Features
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.

https://wiki.ubuntu.com/Security/Features
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](https://wiki.ubuntu.com/Security/Features) for regression tests.*
Pointer Obfuscation

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

See test-glibc-security.py for regression tests.
Address Space Layout Randomization

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.

- http://lwn.net/Articles/184734/

https://wiki.ubuntu.com/Security/Features
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).

https://wiki.ubuntu.com/Security/Features
Exec ASLR

Each execution of a program that has been built with "-ffPIE -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).

https://wiki.ubuntu.com/Security/Features
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).

https://wiki.ubuntu.com/Security/Features
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 should only be 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 x86_64 does not have the same penalties, and will eventually be made the default, but more testing is required.

https://wiki.ubuntu.com/Security/Features
Built with Fortify Source

Programs built with "-D_FORTIFY_SOURCE=2" (and -O1 or higher), enable several compile-time and run-time protection against string attacks 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.

https://wiki.ubuntu.com/Security/Features
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.

https://wiki.ubuntu.com/Security/Features
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).

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

https://wiki.ubuntu.com/Security/Features
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.

https://wiki.ubuntu.com/Security/Features
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.

https://wiki.ubuntu.com/Security/Features
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.

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

https://wiki.ubuntu.com/Security/Features
Kernel Hardening

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

https://wiki.ubuntu.com/Security/Features
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.

https://wiki.ubuntu.com/Security/Features
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, and thus effectively disabled.

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

https://wiki.ubuntu.com/Security/Features
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`. It requires a complete reboot to change this setting.

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

https://wiki.ubuntu.com/Security/Features
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.

https://wiki.ubuntu.com/Security/Features
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/RX

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.

https://wiki.ubuntu.com/Security/Features
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/vmlinux*, /boot/System.map*, /sys/kernel/debug/, /proc/slabinfo

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

https://wiki.ubuntu.com/Security/Features
Blacklist 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 blacklisted 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 specifically loaded via modprobe, or the /etc/modprobe.d/blacklist-rare-network.conf file can be updated to remove the blacklist 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.
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.

https://wiki.ubuntu.com/Security/Features
The key takeaway is that we have thought about all of this for you... so that you don’t have to!
Thank you!

Michael Wood
Sales Executive, Public Sector
michael.wood@canonical.com
978-835-9204

Terry Husbands
Sales Engineer
terry.husbands@canonical.com
813-610-0872