2

LINUX AS A FORENSIC ACQUISITION PLATFORM

This chapter describes Linux as a platform for performing digital forensic acquisition and discusses its various advantages and drawbacks. I also examine the acceptance of Linux and open source software within the digital forensics community, and the final section provides an overview of the relevant Linux fundamentals you'll need to understand subsequent sections of this book.

The examples shown in this book primarily use Ubuntu Linux Server version 16.04 LTS (supported until April 2021) with the Bourne Again shell (Bash), version 4.3.*x*. The examples should also work on other Linux distributions and other OSes, such as OS X or Windows, as long as you use the same or newer tool versions and adjust the device names. Throughout this book, the words *command line*, *shell*, and *Bash* are used interchangeably.

Linux and OSS in a Forensic Context

The growing popularity of *open source software (OSS)* like Linux has made it important as a platform for performing digital forensics. Many researchers have discussed the advantages of using OSS for satisfying the Daubert guidelines for evidential reliability.[1] Brian Carrier, author of Sleuth Kit, explored the legal arguments for using open source forensic tools and suggested that parts of forensic software (but not necessarily all) should be made open source.[2]

The primary advantage of using OSS in a forensic context is transparency. Unlike proprietary commercial software, the source code can be reviewed and openly validated. In addition, academic researchers can study it and build on the work of others in the community. Open source forensic software applications have become the tools and building blocks of forensic science research. There are also disadvantages to using OSS and situations where its use doesn't make sense. In particular, the openness of the open source community may in some cases conflict with the confidential nature of ongoing forensic investigations. Both the advantages and disadvantages of Linux and OSS are discussed in the following sections.

Advantages of Linux and OSS in Forensics Labs

The public availability of OSS means it is accessible to everyone. It is not restricted to those who have purchased licenses or signed nondisclosure agreements. OSS is freely available for download, use, examination, and modification by anyone interested, and no licensing fees or usage costs are involved.

Having access to the source code allows you to customize and facilitate integration with other software, hardware, and processes in a forensic lab. This source-level access increases the possibilities for automating and scripting workloads. Automation reduces the amount of human interaction needed, which limits the risk of human error and frees up these human resources so they can be used elsewhere.

Automation is essential in labs with high volumes of casework to foster optimization and process streamlining. Because you can freely modify the source code, OSS can be customized to meet the requirements of a particular forensic lab. Command line software especially allows you to link multiple tasks and jobs in pipelines with shell scripts to complete an end-to-end process.

Support for OSS has several advantages. The ad hoc community support can be excellent, and mailing lists and chat forums can answer calls for help within minutes. In some cases, quick implementation of patches, bug fixes, and feature requests can occur.

1. Erin Kenneally, "Gatekeeping Out of the Box: Open Source Software as a Mechanism to Assess Reliability for Digital Evidence," *Virginia Journal of Law and Technology* 6, no. 13 (2001).
2. Brian Carrier, "Open Source Digital Forensic Tools: The Legal Argument" [technical report] (Atstake Inc., October 2002).

Linux and OSS are ideal for an academic forensic lab setting, because they use open, published standards rather than closed or proprietary standards. OSS development communities work *with* competing groups instead of against them. Learning from others, copying code and ideas from others (with due attribution), and building on the work of others are encouraged and are the basis for learning and gaining knowledge.

The vendor independence that OSS offers prevents vendor product lock-in and fosters interoperability and compatibility between technologies and organizations. This makes it easier to change the software over time, because individual components can be swapped out with new or alternative technologies without affecting the systems and processes as a whole.

Disadvantages of Linux and OSS in Forensics Labs

The disadvantages of Linux and OSS provide arguments in support of closed proprietary software. Commercial tool implementations often provide benefits and advantages in this area.

The open source community support model is not guaranteed to be reliable, accurate, or trustworthy. The quality of the answers provided by the community can vary greatly; some answers are excellent, whereas others might be wrong or even dangerous. Often no formal support organization exists to help. In situations in which 24/7 support must be guaranteed, commercial providers have an advantage.

Support in the open source world is as transparent as the software, visible for all to see. However, in a forensic lab setting, casework and investigations may be sensitive or confidential. Reaching out to the public for support could reveal or compromise details of an ongoing investigation. Therefore, information security and privacy are issues in the open source support model.

Interoperability with proprietary technology poses difficulties with open source interfaces and APIs. Proprietary technologies that are not public are often reverse engineered, not licensed. Reverse engineering efforts are often incomplete, are at risk of incorrectly implementing a particular technology, and may take a long time to implement.

Free OSS is often a volunteer development effort, and software may be in a perpetual state of development. Some projects may be abandoned or die from neglect. Other projects may experience *forks* in the code where some developers decide to copy an existing code base and take it in a different direction from the original developers.

Free OSS can be rough around the edges. It may be buggy or difficult to learn or use. It may be poorly documented (the source code might be the only documentation). Unlike with commercial software, usually no training is provided with the software product. It takes time and effort to learn Unix/Linux; in particular, the command line is not as intuitive as an all-GUI environment. Many experience a learning curve when they first enter the free, open source world, not just for the software but also for the general attitude and mind-set of the surrounding community.

Commercial software vendors in the forensics community provide a certain degree of defensibility and guarantees for the proper functioning of their software. Some forensic companies have even offered to testify in court to defend the results provided by their software products. In the free, open source community, no one is accountable or will take responsibility for the software produced. It is provided "as is" and "use at your own risk."

Clearly, OSS is not appropriate for every situation, and that is not implied in this book. In many of the examples throughout, OSS is more useful for educational purposes and to show how things work than it is a viable alternative to professional commercial forensic software.

Linux Kernel and Storage Devices

Traditional Unix systems, from which Linux inherits its philosophy, were designed in a way that everything on them is a file. Each file is designated as a specific type, which includes regular files and directories, block devices, character devices, named pipes, hard links, and soft/symbolic links (similar to LNK files in Windows). On the examiner workstation, files of interest to forensic investigators are the block device files of attached subject disks that potentially contain forensic evidence. This section describes Linux devices—in particular, block devices for storage media.

Kernel Device Detection

Unix and Linux systems have a special directory called */dev*, which stores special files that correspond to devices understood by the kernel. Original Unix and Linux systems required manual creation of device files in the */dev* directory (using the `mknod` command) or had scripts (`MAKEDEV`) to create devices on boot or when required. With the arrival of plug-and-play hardware, a more dynamic approach was needed, and `devfs` was created to automatically detect new hardware and create device files. The requirement to interact better with userspace scripts and programs led to the development of `udev`, which replaced `devfs`. Today, `udev` has been merged into `systemd` and runs a daemon called `systemd-udevd`.

When a new device is attached to (or removed from) a host, an interrupt notifies the kernel of a hardware change. The kernel informs the `udev` system, which creates appropriate devices with proper permissions, executes setup (or removal) scripts and programs, and sends messages to other daemons (via `dbus`, for example).

To observe `udev` in action, use the `udevadm` tool in monitor mode:

```
# udevadm monitor
monitor will print the received events for:
UDEV - the event that udev sends out after rule processing
KERNEL - the kernel uevent

KERNEL[7661.685727] add      /devices/pci0000:00/0000:00:14.0/usb1/1-14 (usb)
KERNEL[7661.686030] add      /devices/pci0000:00/0000:00:14.0/usb1/1-14/1-14:1.0
```

```
    (usb)
KERNEL[7661.686236] add      /devices/pci0000:00/0000:00:14.0/usb1/1-14/1-14:1.0/
    host9 (scsi)
KERNEL[7661.686286] add      /devices/pci0000:00/0000:00:14.0/usb1/1-14/1-14:1.0/
    host9/scsi_host/host9 (scsi_host)
...
KERNEL[7671.797640] add      /devices/pci0000:00/0000:00:14.0/usb1/1-14/1-14:1.0/
    host9/target9:0:0/9:0:0:0/block/sdf (block)
KERNEL[7671.797721] add      /devices/pci0000:00/0000:00:14.0/usb1/1-14/1-14:1.0/
    host9/target9:0:0/9:0:0:0/block/sdf/sdf1 (block)
...
```

Here a disk has been plugged into a USB port, and `udev` has managed the setup of all the appropriate device files and links.

The `udevadm` command can also be used to determine a list of the associated files and paths for attached devices. For example:

```
# udevadm info /dev/sdf
P: /devices/pci0000:00/0000:00:14.0/usb1/1-14/1-14:1.0/host9/target9:0:0/9:0:0:0/
    block/sdf
N: sdf
S: disk/by-id/ata-ST2000DL003-9VT166_5YD83QVW
S: disk/by-id/wwn-0x5000c50048d79a82
S: disk/by-path/pci-0000:00:14.0-usb-0:14:1.0-scsi-0:0:0:0
E: DEVLINKS=/dev/disk/by-path/pci-0000:00:14.0-usb-0:14:1.0-scsi-0:0:0:0 /dev/disk/
    by-id/wwn-0x5000c50048d79a82 /dev/disk/by-id/ata-ST2000DL003-9VT166_5YD83QVW
E: DEVNAME=/dev/sdf
E: DEVPATH=/devices/pci0000:00/0000:00:14.0/usb1/1-14/1-14:1.0/host9/target9:0:0/
    9:0:0:0/block/sdf
E: DEVTYPE=disk
E: ID_ATA=1
...
```

Understanding the Linux device tree is important when you're performing forensic acquisition and analysis activities. Knowing which devices are part of a local investigator's machine, which devices are the suspect drives, which device is the write blocker, and so on is crucial when you're running forensic commands and collecting information from a device.

Storage Devices in /dev

Attached drives will appear as block devices in the */dev* directory when they're detected by the kernel. Raw disk device files have a specific naming convention: *sd** for SCSI and SATA, *hd** for IDE, *md** for RAID arrays, *nvme*n** for NVME drives, and other names for less common or proprietary disk device drivers.

Individual partitions discovered by the kernel are represented by numbered raw devices (for example, *hda1*, *hda2*, *sda1*, *sda2*, and so forth).

Partition block devices represent entire partitions as a contiguous sequence of disk sectors. A partition typically contains a filesystem, which can be mounted by the kernel and made available to users as a normal part of the directory tree. Most forensic tools can (and should) examine raw devices and partition devices without having to mount the filesystem.

Other Special Devices

Several other devices are useful to know for the examples in this book. The bit bucket, */dev/null*, discards any data written to it. A steady stream of zeros is provided when accessing */dev/zero*. The random number generator, */dev/random*, provides a stream of random data when accessed. Tape drives typically start with */dev/st*, and you can access other external media via */dev/cdrom* or */dev/dvd* (these are often symbolic links to */dev/sr**). In some cases, devices are accessed through the generic SCSI device driver interface */dev/sg**.

Other special pseudo devices include */dev/loop** and */dev/mapper/** devices. These devices are discussed in more detail throughout the book.

Linux Kernel and Filesystems

Filesystems organize storage into a hierarchical structure of directories (folders) and files. They provide a layer of abstraction above the block devices.

Kernel Filesystem Support

The Linux kernel supports a large number of filesystems (for a list, see *https://en.wikipedia.org/wiki/Category:Linux_kernel-supported_file_systems*), which can be useful when performing some forensics tasks. However, filesystem support is not necessary when performing forensic acquisition, because the imaging process is operating on the block device below the filesystem and partition scheme.

To provide a consistent interface for different types of filesystems, the Linux kernel implements a Virtual File System (VFS) abstraction layer. This allows mounting of regular storage media filesystems (EXT*, NTFS, FAT, and so on), network-based filesystems (nfs, sambafs/smbfs, and so on), userspace filesystems based on FUSE,[3] stackable filesystems (encryptfs, unionfs, and so on), and other special pseudo filesystems (sysfs, proc, and so on).

The Linux Storage Stack Diagram, shown in Figure 2-1, helps you understand the relationship among filesystems, devices, device drivers, and hardware devices within the Linux kernel.

3. FUSE is a userspace filesystem implementation (see *https://en.wikipedia.org/wiki/Filesystem_in_Userspace*).

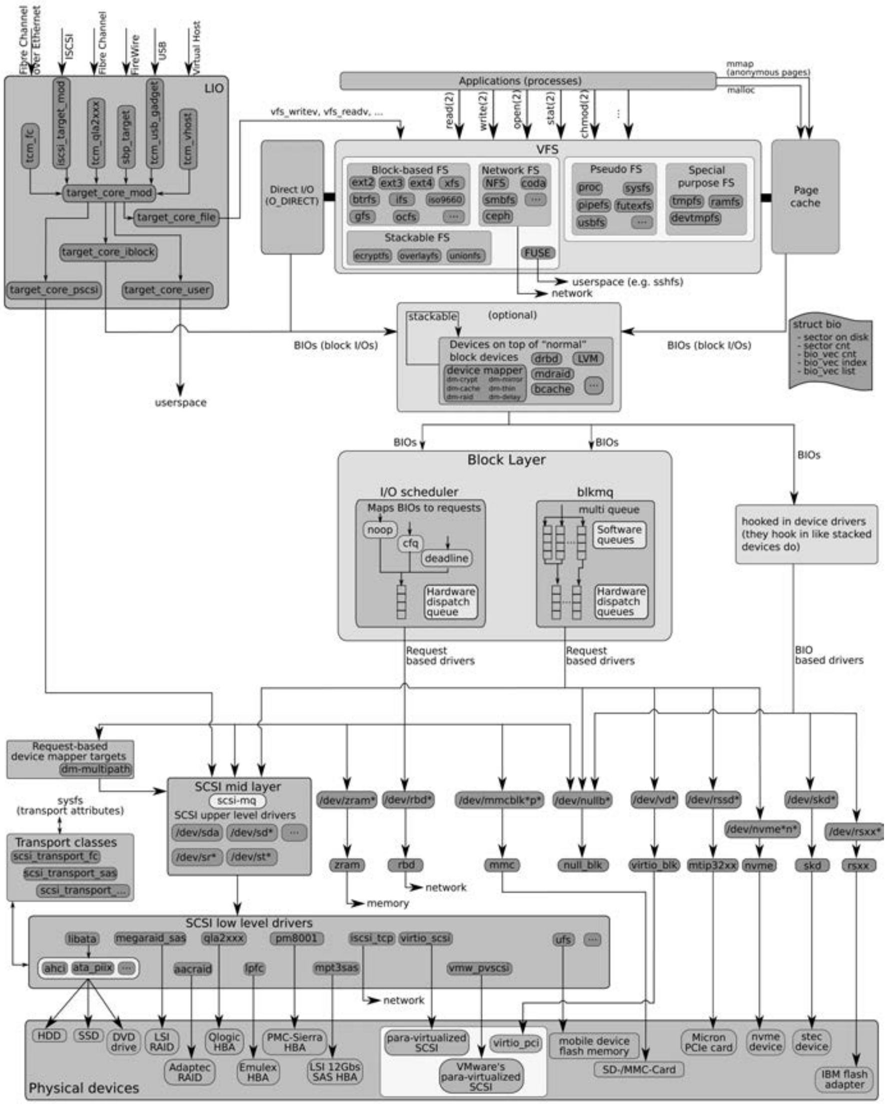
Figure 2-1: The Linux Storage Stack Diagram (https://www.thomas-krenn.com/en/wiki/Linux_Storage_Stack_Diagram, used under CC Attribution-ShareAlike 3.0 Unported)

Mounting Filesystems in Linux

An often-misunderstood concept is the difference between an attached disk device and a mounted disk device. A device does not need to be mounted to acquire it or even to access it with forensic analysis tools. Forensic tools that operate directly on block devices will have access to attached disks without mounting them through the OS.

Filesystems that reside on disk devices in Unix and Linux require explicit mounting before being accessible as a regular directory structure. *Mounting* a filesystem simply means it is made available to use with standard file access tools (file managers, applications, and so on), similar to drive letters in the DOS/Windows world. Linux doesn't use drive letters; mounted disks become part of the local filesystem and are attached to any chosen part of the filesystem tree. This is called the filesystem's *mount point*. For example, the following command mounts a USB stick on an investigator system using (*/mnt*) as the mount point:

```
# mount /dev/sdb1 /mnt
```

To physically remove a mounted disk in Linux, unmount the filesystem first to prevent corruption of the filesystem. You can use the `umount` command (that is `umount`, not `unmount`) with either the device name or the mount point. These two commands perform the same action to unmount a disk filesystem:

```
# umount /dev/sdb1
# umount /mnt
```

After the filesystem is unmounted, the raw disk is still visible to the kernel and accessible by block device tools, even though the filesystem is not mounted. An unmounted disk is safe to physically detach from an investigator's acquisition system.

Don't attach or mount suspect drives without a write blocker. There is a high risk of modifying, damaging, and destroying digital evidence. Modern OSes will update the last-accessed timestamps as the files and directories are accessed. Any userspace daemons (search indexers, thumbnail generators, and so on) might write to the disk and overwrite evidence, filesystems might attempt repairs, journaling filesystems might write out journal data, and other human accidents might occur. You can mount a filesystem while using a write blocker, and it will be accessible in the same way as a regular filesystem but in a read-only state, ensuring digital evidence is protected.

Accessing Filesystems with Forensic Tools

When you're using forensic tools, such as Sleuth Kit, dcfldd, foremost, and others, you can access the filesystem (without mounting) by using the correct block device representing the partition where the filesystem resides. In most cases, this will be a numbered device, such as */dev/sda1*, */dev/sda2*, or */dev/sdb1*, and so on, as detected by the Linux kernel.

In cases where the Linux kernel does not detect the filesystem, you may need to explicitly specify it. A filesystem will not be correctly detected for any of the following reasons:

- The filesystem is not supported by the host system (missing kernel module or unsupported filesystem).
- The partition table is corrupted or missing.
- The partition has been deleted.
- The filesystem offset on the disk is unknown.
- The filesystem needs to be made accessible (unlock device, decrypt partition, and so on).

In later sections of the book, I'll explain techniques that use loop devices to access partitions and filesystems that are not automatically detected by the Linux kernel or various forensic tools.

Linux Distributions and Shells

When you're creating an investigator workstation to perform digital forensic acquisition or analysis work, it's useful to understand the basic construction or composition of a Linux system.

Linux Distributions

The term *Linux* technically refers only to the *kernel*, which is the actual OS.[4] The graphical interface, tools and utilities, and even the command line shell are not Linux but parts of a Linux *distribution*. A distribution is a functional package that typically contains the Linux kernel, installers and package managers (usually unique to the distribution), and various additional programs and utilities (including standard applications, such as Office suites, web browsers, or email/chat clients). There is only one official Linux kernel, but there are many Linux distributions—for example, Red Hat, SUSE, Arch, and Debian, among others. There are also many derivative distributions. For example, Ubuntu is a derivative based on Debian, CentOS is based on Red Hat, and Manjaro is based on Arch. For a comprehensive list of distributions (and other non-Linux, open source OSes), visit *http://distrowatch.com/*.

Multiple components make up the graphic interface of various Linux distributions and are useful to understand. The X11 window system is a display server that interacts with the graphics hardware and provides an interface to the X11 graphics primitives (Wayland is a newer alternative to X11). A window manager controls movement, resizing, placement, and other windows management on a system. Some examples of window managers

4. There is some naming controversy regarding the inclusion of *GNU* with *Linux*, see *https://en.wikipedia.org/wiki/GNU/Linux_naming_controversy*.

include Compiz, Mutter, and OpenBox, and you can use them without a desktop environment. Desktop environments provide the look and feel of a distribution and operate on top of the window manager. Examples of popular desktops are Gnome, KDE, Xfce, and Mate. The graphics environment you choose for your forensic investigator's workstation can be based on your personal preference; it doesn't have any impact on the evidence you collect or analyze. The examples shown in this book were performed on a system without a GUI (Ubuntu Server version).

The Shell

The shell is a command prompt that humans and/or machines use to submit commands to instruct and control an OS. The shell starts or stops programs, installs software, shuts down a system, and performs other work. Arguably, the command shell offers more powerful features and possibilities than graphical environments.

The examples in this book use the command line environment. Some GUI equivalents or GUI frontends to the command line tools may exist, but they are not covered in this book.

The most common shell in use today, and the default in most Linux distributions, is Bash. The examples in this book use Bash but may also work on other shells (zsh, csh, and so on).

Command Execution

The shell is simply another program that runs on a system. Human users interface with it in the form of typed commands, and machines interface with it in the form of executed shell scripts.

When human users enter commands, they usually type them into the prompt and then press ENTER or RETURN. There may or may not be any output, depending on the program run and the configuration of the shell.

Piping and Redirection

A useful feature of the Unix/Linux command line is the ability to pass streams of data to programs and files using piping and redirection. This is somewhat similar to drag-and-drop and copy/paste in graphical environments, but with much more flexibility.

A program can receive data from the output of other programs or from files on the filesystem. A program can also output data to the input of another program or send it to a file on the filesystem.

The following examples illustrate *tool.sh* redirecting output into *file.txt*, receiving input from *file.txt*, and piping output from *tool.sh* to the input of *othertool.sh*:

```
$ tool.sh > file.txt
$ tool.sh < file.txt
$ tool.sh | othertool.sh
```

This piping and redirection mechanism is not limited to single commands or files and can be chained in a sequence with multiple programs:

```
$ tool.sh < file.txt | othertool.sh | lasttool.sh > lastfile.txt
```

Pipelines and redirection are used extensively throughout this book. They allow you to complete multiple tasks using a single line of commands, and they facilitate scripting and automation, eliminating the need for human interaction. The examples in this book use piping and redirection to acquire images of storage media, move data between forensic programs, and save evidential information of interest in files.

Closing Thoughts

In this chapter, I discussed the use of Linux as a viable platform to perform forensic acquisition tasks and covered both its advantages and disadvantages. I provided a review of Linux distributions and how the Linux kernel works. I showed the concept of devices and filesystems and the use of shells, piping, and redirection from the perspective of the forensic examiner. You now have the Linux knowledge needed to understand the examples in the rest of the book.