Linux investigations - USB devices and keyboard layouts.

Taz Wake | Halkyn Consulting Ltd | Originally published 2025-02-21

During investigations, from insider threat cases to nation-state intrusions, we often need to understand how a device is configured, who has used it, and so on. In this article, we will look at some of the steps you can take to do this when investigating a Linux-based operating system.

As a quick caveat, this article will **not** be an exhaustive list of things you can (or should) check. Instead, I wanted to highlight some of the things that have been useful for my investigations recently and which often get overlooked during the hectic incident response cycle. Also, this material is really aimed at supplementing the content taught on SANS FOR577 (https://sans.org/for577), so I will assume that level of knowledge.

Attaching Devices

Often a starting point for investigations - we need to understand if a user has attached anything *weird* to the device, or for insider cases, are they exfiltrating data over a USB?

What USB devices have been attached?

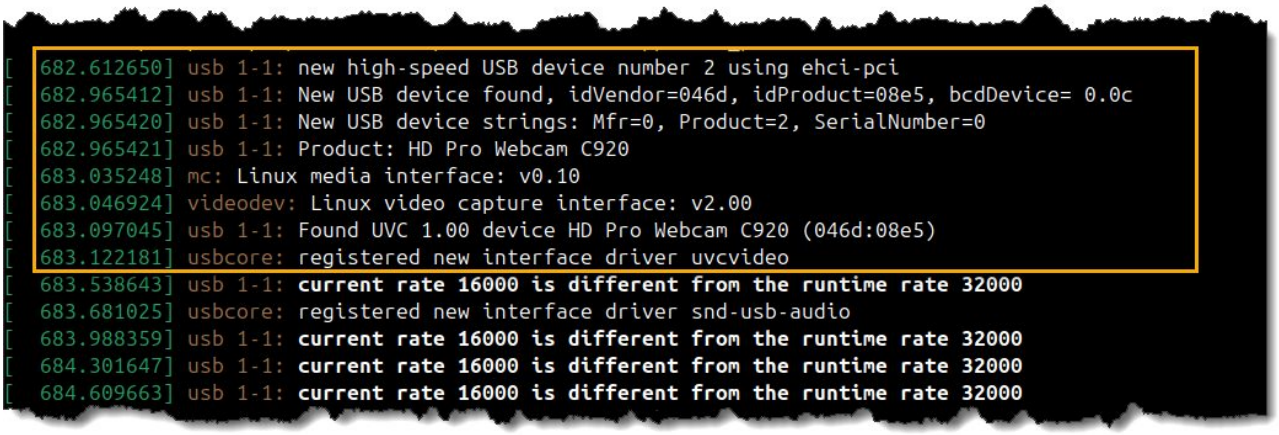
We often need to check what devices have been attached to a system, and really any USB device falls into this category. Unfortunately, this isn't as simple a task as it can be when investigating Windows.

The first thing to review is the kernel message logs or the journal. In some distro versions, this will be in /var/log/kern.log, in some it will be /var/log/dmesg, or in others (RHEL 7+, Ubuntu 24.04+) it will be in the Journal logs.

When a device is attached, the kernel will register this as a "new device." The exact syntax can vary but it gives us some keywords to search for. Below are some examples of how this can appear in the data.

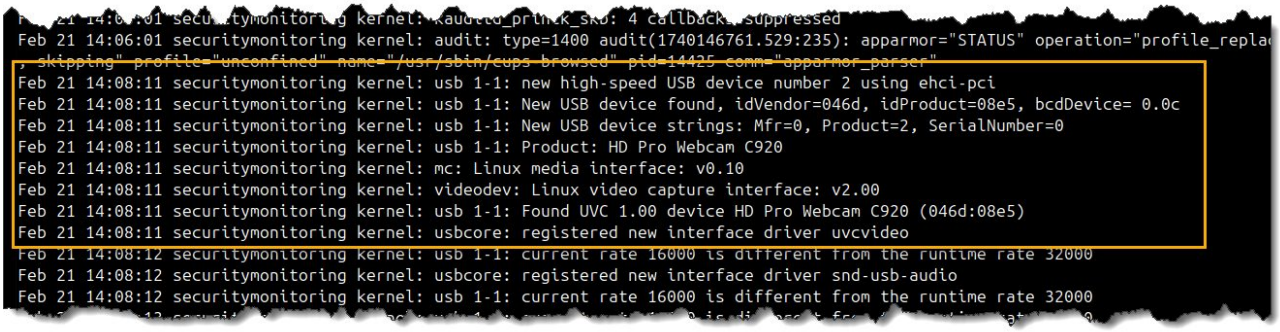
Looking for a USB camera on a live system

- In this first example, we can simply use the dmesg command and scroll a lot.



Screenshot with dmesg output showing entries relating to the C920 webcam being attached.

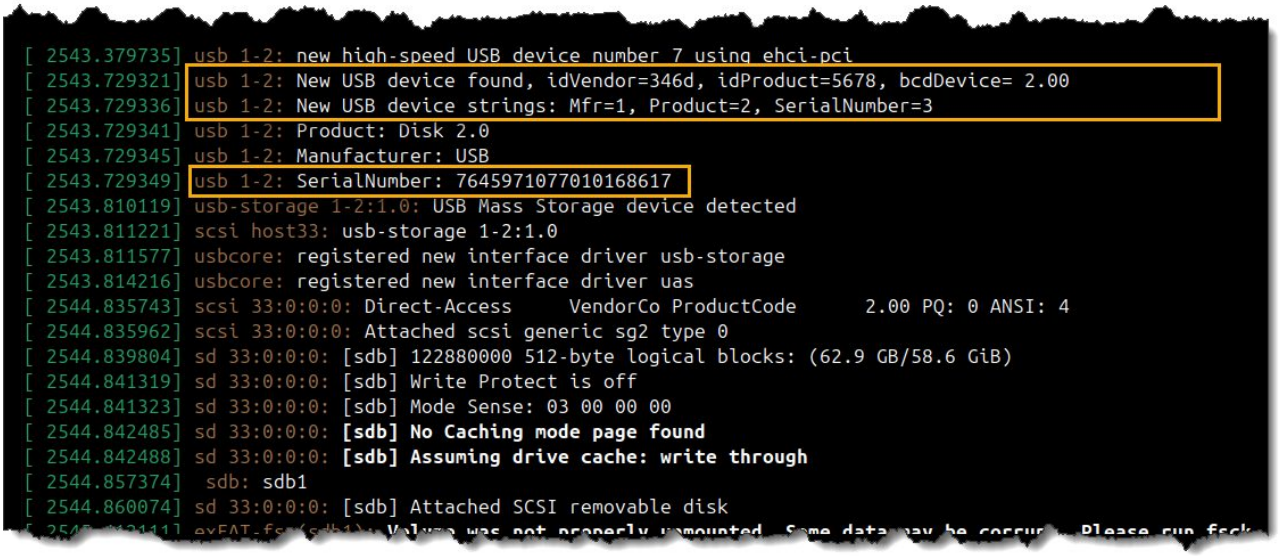
- On live systems, another approach is to use the Journal. This can make it easier to read data (especially the timestamps) but is a lot noisier. You really need to filter this. A good example is:



Screenshot with journalctl output, only showing kernel entries and focused on USB activity.

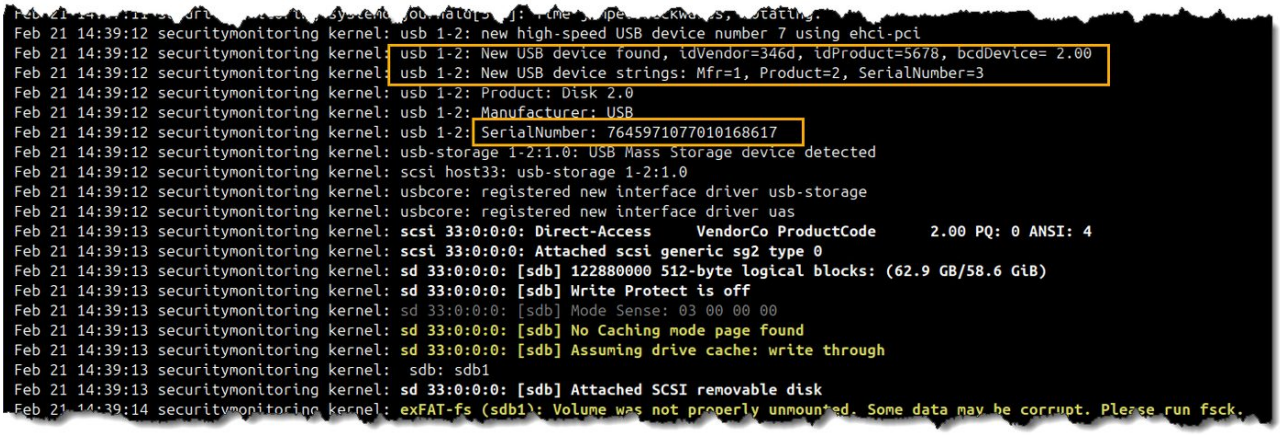
Looking for a removable storage device on a live system

The commands are the same but the output becomes a bit more interesting. Here is an example of dmesg output



Screenshot showing dmesg output, looking at an attached USB storage device.

As you'd expect, the Journal shows the same data



Screenshot showing journalctl output detailing the attached USB device.

Now we can see the attached devices, there are some key fields to be in interested in:

usb 1-2: This is the bus and port number for the device. In the examples above, both are on USB bus 1. The camera has gone to port 1, the USB storage device has gone onto port 2.

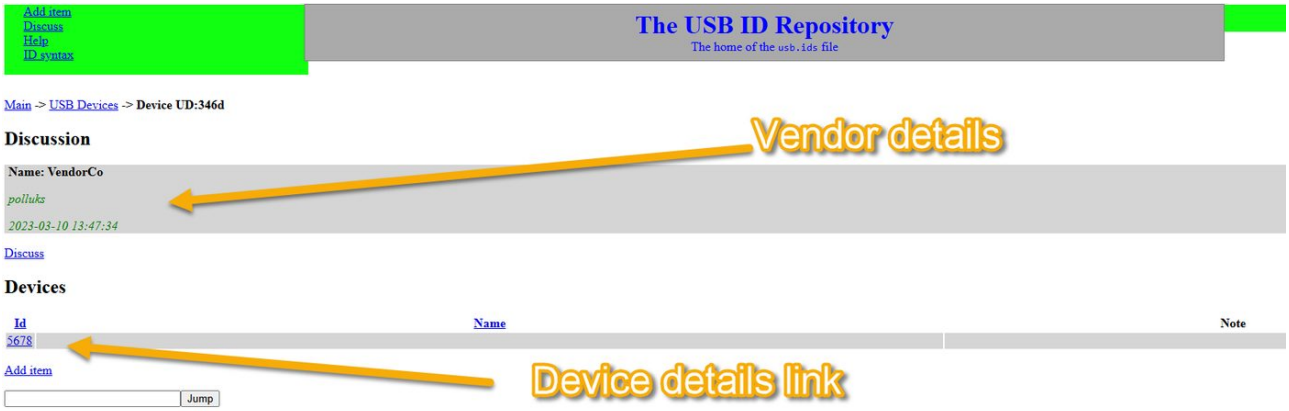
idVendor: This field contains the vendor ID for the device. This can (sometimes) be found in your local system at /usr/share/misc/usb.ids but this is often out of date. In the examples above, the 046d does show up, however the 346d doesn't.

```
root@securitymonitoring:~# grep ^046d /usr/share/misc/usb.ids
046d  Logitech, Inc.

root@securitymonitoring:~# grep ^346d /usr/share/misc/usb.ids
root@securitymonitoring:~#
root@securitymonitoring:~#
```

Searching for USB vendor IDs in local data.

An alternative approach is to use http://www.linux-usb.org/usb-ids.html which can be more relevant. In this example, it does identify the USB device (which was a cheap, no brand device bought on Amazon)



The online data from the USB ID repository

idProduct: This is a product ID assigned by the manufacturer. It is occasionally possible to look for this - for example, Logitech does show the product ID matches a C920 Pro HD Webcam. However, not all vendors are this diligent and the cheap, no-brand, USB storage device has no data here.

bcdDevice: The binary coded decimal device version. This is an internal reference from the device.

Serial Number: Some USB devices, most often storage devices, will report a serial number to the kernel. If so, this is recorded here.

Further investigations

You can use the **lsusb** command in Linux to understand more about a specific USB device. This is the "list USB devices" command and is a utility for displaying information about the buses and connected devices.

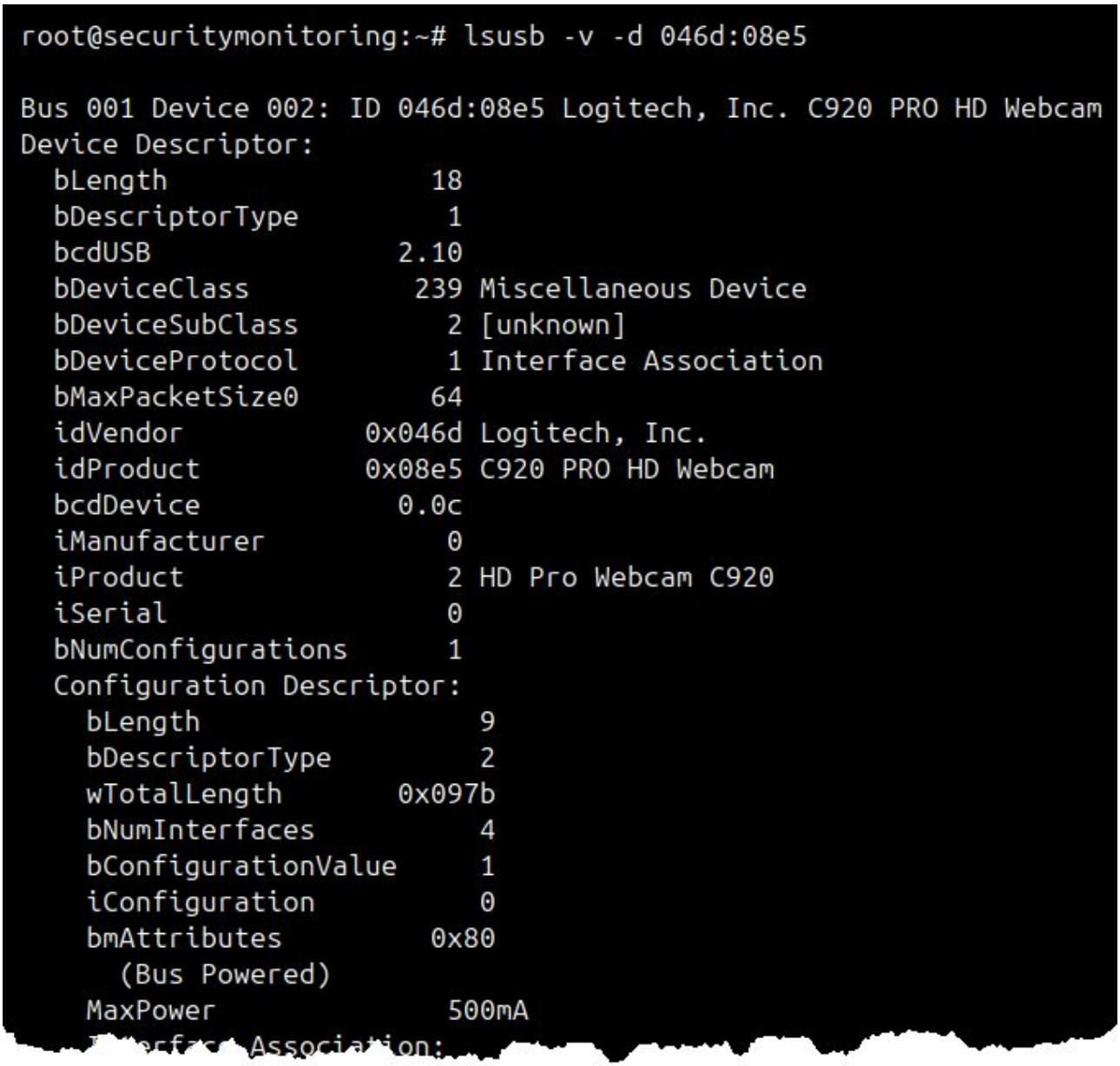
It can provide some useful detail for live response, but be warned, it can also be very, very noisy. The example syntax is:

```
```

This is an example of the output on the USB drive:

Truncated view of the output from lsusb pointed to the USB Storage device.

This is the output of lsusb run against the webcam - it contains a lot more information



Truncated view of the output from lsusb pointed to the USB webcam.

To give an example of the variation in detail, the USB storage device returned 68 lines of output, while the webcam returned 1388 lines. This is almost entirely down to how the manufacturer configures the device, what data they store and what is made available.

There isn't a *lot* of consistency between devices and vendors.

Physical keyboard layout

This is a good way to learn about the user and gives a strong clue about language or regionalisation settings on the system. There are some challenges here, especially with extensively multi-user devices, and remember lots of people default to the US English keyboard layout. Even if it isn't the user's native layout, it is sometimes easier to keep the US English setting, rather than change it.

Live Response Checks

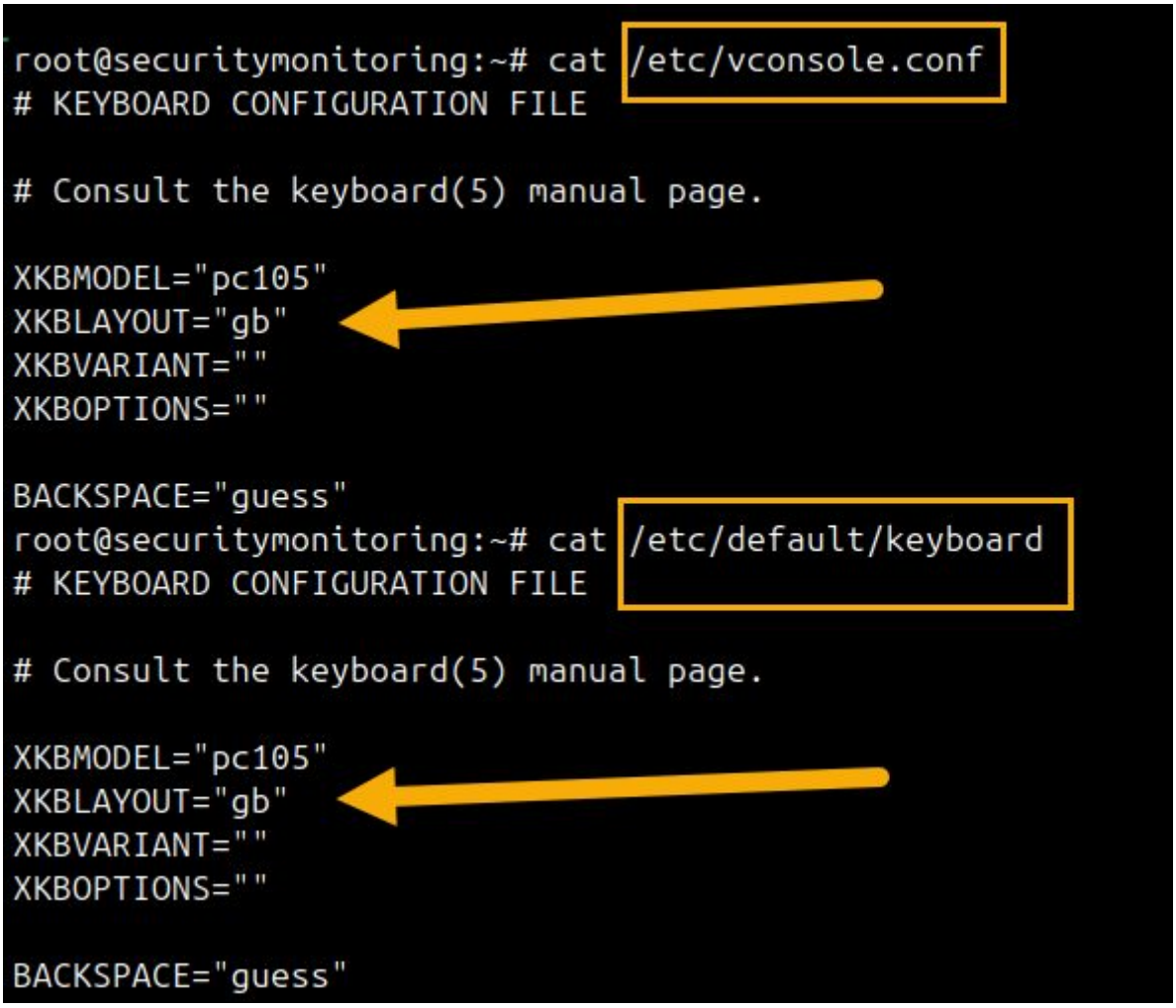
The easiest approach in live response is to simply hit a few characters on the keyboard, this will give you a good idea of the layout.

However, during an investigation, we need better details. This means we need to check the configuration files.

The keyboard layout is stored at multiple locations:

- /etc/vconsole.conf (this configures the virtual consoles or tty devices)
- /etc/default/keyboard (this is the system-wide keyboard setting for all users in X11)
- Window Manager Specific locations (this can vary significantly)

The easiest two to check are the default keyboard or vconsole configuration files, and they generally hold the same data.



Screenshot showing the contents of two configuration files which store data on keyboard layouts.

It is a little bit more complicated to check the window manager details as each manager has a different approach. The two most common are:

GNOME: The user's keyboard layout can be stored in either a **dconf** database or **gsettings**. There may need to be an element of trial and error here and you could discover one of three states (or more) is true:

1. The keyboard layout is being managed by the system-wide X11 configuration, where /etc/default/keyboard is used.
2. The keyboard layout is being managed by **gsettings**. You can validate this by running **gsettings get org.gnome.desktop.input-sources sources**. If you get meaningful output, such as [('xkb','gb')] then you know gsettings is working.
3. The keyboard layout is being managed by a **dconf** database. This could store the keyboard data in multiple locations - for example **org.gnome.libgnomekbgd** or **org.gnome.desktop.input-sources**. The exact choice varies by distro and age (libgnomekbgd seems to be a bit older, but YMMV). You can query this by running a command similar to **dconf read /org/gnome/libgnomekbgd** or **dconf read /org/gnome/desktop/input-sources/sources**. There may need to be trial and error to the specific syntax for your environment.

KDE Plasma: This is arguably much easier. Inside the user's home directory, there should be a file at ~/.config/kxkbrc. This stores the keyboard layout on a per-user basis, while the system-wide configurations are at /etc/default/keyboard.

Other ways to check

For Wayland users (the "modern" replacement for the ageing X11 protocol) there are some additional live response commands available.

```
root@securitymonitoring:~# localectl status
System Locale: LANG=en_US.UTF-8
    VC Keymap: (unset)
   X11 Layout: gb
    X11 Model: pc105
```

Screenshot showing the output of localectl status.

And for Wayland and X11 systems, you can use

```
root@securitymonitoring:~# setxkbmap -query
rules:      evdev
model:      pc105+inet
layout:     gb,us
variant:    ,
```

Screenshot showing the output of setxkbmap -query. Note this shows all the layouts available, not only the active keyboard.

What about captured evidence?

In both the previous sections, we've looked entirely at how to find information out during live response. This is great, and we should try to push as much of our IR to live-response as possible, but it isn't always an option.

What happens if you are given a full disk image as an E01 or RAW image, for example?

The good news is that most of the data is stored in text files, so we can just query them. This is where your IR skills become useful, but the basic choices are to mount the image and investigate or use a tool like the TSK (or dissect) to directly query the evidence.

```
taz@Glaive:/mnt/d/Cases/caseid5778222/diskimages$ fcat -o 1054720 /etc/default/keyboard Webserver_20250213.E01
# KEYBOARD CONFIGURATION FILE

# Consult the keyboard(5) manual page.

XKBMODEL="pc105"
XKBLAYOUT="gb"
XKBVARIANT=""
XKBOPTIONS=""

BACKSPACE="guess"
```

Screenshot showing TSK being used to query the /etc/default/keyboard file inside an E01 image.

When it comes to log data, keep in mind this can be stored in different locations.

- dmesg. Since RHEL version 7 and Ubuntu 24.04, the dmesg output is no longer written to /var/log/dmesg (or /var/log/kern.log). Instead, the data is really only memory resident or written to the journal via the Kernel message facility. If you have an older system, or one that has been upgraded but maintains the traditional syslog data, this will be in **/var/log** on the target system.
- Journal data. This is the current standard way to store all system logs in RHEL and Ubuntu family Linux distros. The journal writes log data to **/var/log/journal/[UUID]/[uid]@[logon session].journal** files. These are binary files and best read with the journalctl command.

Summary

This is just a taste of the information available to an investigator looking at Linux systems. Hopefully, it will give you ideas about what else you can find during an incident or user-investigation and I will try to add more articles in the future.

The good news is that Linux stores a lot of its important configuration data in text files, so we get good visibility with evidence as well as during live response, but the best option is always live!

Some useful commands include (but aren't limited to):

- dmesg
- journalctl
- lsusb
- localectl

- setxkbmap

I hope this helps a little bit with your future investigations and happy hunting!

About Halkyn Consulting

Halkyn Consulting Ltd is a UK-based security and risk management consultancy providing 24/7 incident response, digital forensics, threat hunting, virtual security leadership (vCISO) and security compliance support to organisations worldwide. We deliver clear, practical, vendor-independent advice based on real-world threats — no hype, no upsell.

More free security resources are available at www.halkynconsulting.co.uk/security-resources.

Web: www.halkynconsulting.co.uk | **Email:** info@halkynconsulting.co.uk | **Tel:** +44 1244 940858