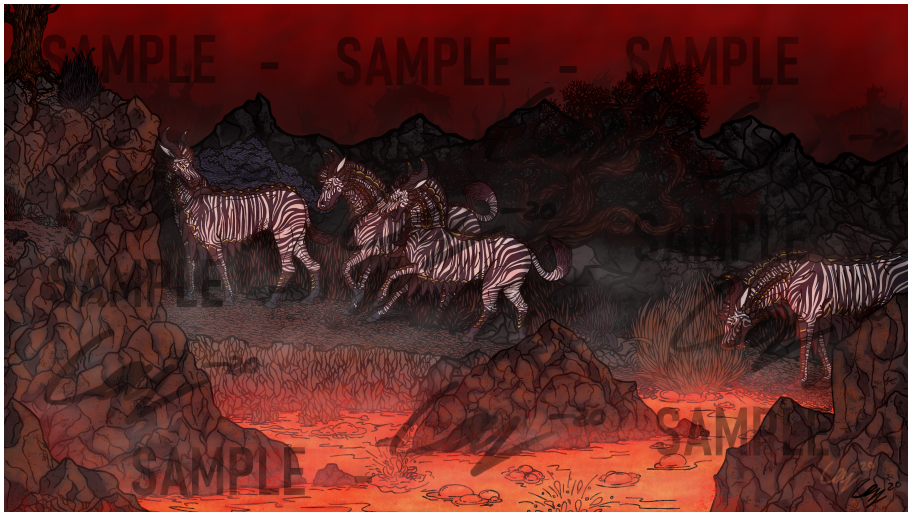
How To Deal With Your Raspberry Spy

Gavin L. Rebeiro

March 2, 2021

Cover



- Artist: Cay
- Artist Contact Details: catthecay@gmail.com
- Artist Portfolio: https://thecayart.wixsite.com/artwork/contact

These covers are here to spice things up a bit. Want to have your artwork showcased? Just send an email over to

contact@e2eops.io

and let us know!

For encrypted communications, you can use the OpenPGP Key provided in chapter 6.

Copyright

Chapter 1

Acknowledgements

Techrights techrights.org (TR) deserves credit for coverage of the Raspberry Spy Foundation's underhand tactics; a heartfelt thanks to everyone who participated and notified TR about the Raspberry Spy espionage. TR has been robbed of credit they deserved in the early days of news coverage. The following links go over some of the news coverage from TR:

- Raspberry Pi (at Least Raspbian GNU/Linux and/or Raspberry Pi Foundation) Appears to Have Been Infiltrated by Microsoft and There Are Severe Consequences
- Raspberry Pi Foundation is Trying to Cover Up Its Deal With the Devil by Censoring Its Own Customers
- Raspberry Pi Foundation Owes Customers an Apology
- Holding the Raspberry Pi Foundation Accountable by Explaining What Happened (and Providing Evidence)
- Raspberry Pied in the Face – Part I: What is Known About the Relationship Between Microsoft and the Raspberry Pi Foundation
- Raspberry Pi OS Adds Microsoft Repository Without User Permission
- Raspberry Pied in the Face – Part II: Raspberry Pi Foundation in Violation of GNU/Linux Rules (Because of Microsoft)
- Raspberry Pied in the Face – Part III: Eben Upton's Response and Its Significance

- Raspberry Pied in the Face – Part IV: Poor Crisis Management by the Raspberry Pi Foundation
- Raspberry Pied in the Face – Part V: Raspberry Bye? The Lost of Trust is Pervasive and the (Un)Official Response Unhelpful
- What Microsoft Did to the Raspberry Pi Foundation is Part of a Broader Anti-GNU/Linux Strategy
- More Than a Fortnight After Installing Microsoft Surveillance and Keys on Millions of Computers Without Users' Consent the Spin Comes From the Raspberry Pi Company (via Microsoft)
- Raspberry Pi Reaffirms Its Commitment to Microsoft (as Trojan Horse Inside Classrooms) and Abandons the Free Software Community
- Microsoft Inside – Part II: Microsoft Has Plans for the Raspberry Pi or Linux SBCs in General (and It Hides Its Role in That)

I tried to keep things in chronological order, but you should just check out the TR site archives from Febuary 2021 onwards for coverage on this treachery from the Raspberry Spy Foundation.

A big thanks also to the founder of Everything Wrong With Free Software (EWWFS); EWWFS did a great piece on the Raspberry Spy that is worth reading:

- maybe-dont-buy-a-raspberry-spy

I originally got the inspiration for this paper from the creator of EWWFS. Much support and encouragement was provided during the research and development of this paper from EWWFS.

I saw the freedom-crushing propaganda machine in full action after the story first broke. Every single forum post that was censored, every single person in chat rooms that had been blocked or called a "basher", and every single person who boycotts or called for a boycott of the Raspberry Spy, all give us hope that the fight for freedom is still alive.

It is my hope this document helps you advance your freedoms, learn a few new things, and have fun in the process. Let’s get hacking!

Contents

Chapter 2

Introduction

We don't want to be spied on; what happens when we're faced with an operating system that spies on people? We throw it in the trash where it belongs! I am boycotting the Raspberry Spy myself (you're free to join me in doing so) but I don't want people to waste hardware that they already have. So we're going to walk through an interesting path of installing a different operating system on the Raspberry Spy; I want to show you a few things that will empower you to take greater control over your computing.

We'll gently walk through and explore the following: how to install an operating system on an embedded device (a Raspberry Spy, in this case) over a USB-to-UART bridge (UTUB). This is the main project we've got on our hands. Don't worry if you've never touched embedded systems before; everything here is accessible to people with a modest set of prerequisite knowledge and some basic apparatus.

We'll delve into things with more depth as we move forward with our project; if you don't understand something when you first encounter it, just keep reading.

2.1 Prerequisite Knowledge

There's not much prerequisite knowledge required. Here's what you need to know:

- A basic grasp of how to operate a shell on a GNU/Linux system. GNU Bash is an example. You don't need to know

how to write shell scripts. Knowledge of how to use the shell interactively will suffice.

That's it. Really. Anything else you need you will pick up on the way.

2.2 Apparatus

You will need the following apparatus:

- A Raspberry Spy. I've got the Raspberry Spy Model 3 B+ so that's what I'll be using in this project.
- A working internet connection.
- A USB thumb drive (used as boot media) for the Raspberry Spy.[1]
- A power supply for the Raspberry Spy.
- A USB-to-UART bridge (UTUB). I've got a CP2104 from Silicon Labs; this is widely available and you can pick it up from an online retailer. You want a module that has all the necessary pins and peripherals already packaged into one, neat, unit. I believe the specific module I have is by WINGONEER.
- 3 female-to-female jump wires.
- A computer with any recent GNU/Linux installed on it. The computer needs to have a working USB port.
- A generic microSD card reader/writer. I have an Anker AR200.

It's likely that you already have the apparatus to operate your Raspberry Spy. Just acquire the additional bits that you don't already have. The list here is just for completeness.

Here's some extra equipment that will make your life easier:

[1] If you've got a Raspberry Spy that can only accept an SD card as boot media, you don't need to fret too much. The procedure is the same; you just write the OS image to an SD card instead of a USB thumb drive. Fixing quirks of SD card installations are, however, out of scope of this project; you should refer to the relevant documentation, IRC chats, and mailing lists. I will provide links to boot-media-specific information, when we discuss boot media; this should give you a starting point to troubleshoot issues.

- When you're dealing with electronics, you should heed the old idiom of "two is one and one is none". Get spares of whatever you can, as a rule.
- A digital multimeter (DMM) with spare fuses for the multimeter. Being able to do some quality control (QC) before you hook up your UTUB to your hardware is going to give you peace of mind. Don't skimp on the spare fuses for the DMM; it's easy to forget how much current you've got flowing through a circuit and fry the DMM's fuse by accident[2].
- A 2M or longer USB extension cable. Male-to-female is what you want here. You plug in the male part to your computer and the female part is open for receiving the UTUB. This makes life a lot easier (and safer).
- Nitrile gloves. Helps keep you safe.
- Safety goggles. Again, doesn't hurt to be careful.

You should now have everything you need to get started!

[2] Real fuses were harmed during the making of this document.

Chapter 3

Fundamentals

Now that you know what you need to get started, let's gently walk through an overview of the fundamental ideas and topics that we'll be engaging and experimenting with.

The order of topics may seem strange at first but things should make sense as you move on.

3.1 Communication

If we want two computers to communicate, there needs to be some protocol that they both speak.

If we want two computers to communicate, there needs to be a physical medium over which they can communicate.

Almost all computers and their peripherals communicate over USB these days. But when it comes to getting into the nitty-gritty details, you will often find UART humbly serving the same purpose it has for decades of computing. Fortunately for us, almost every embedded system these days supports UART; this includes the Raspberry Spy.

We'll be using our UTUB to install a new OS on our Raspberry Spy over a serial interface (UART). The program that we'll be using to do this serial communication is picocom(1).

Why bother with this anachronistic technology? Glad you asked! Once you know how to operate something like a UTUB and a program like picocom(1), you can "break into" several

devices and modify them how you wish. Routers, motherboards, embedded systems, etc. all tend to have some sort of serial interface on them. Once you learn the basics, you are equipped to liberate yourself and gain more computing freedom.

But wait. Isn't all this embedded stuff way too difficult and only for "experts"? HOGWASH! You can do it too. Don't fall for the propaganda. You are perfectly capable of doing a bit of serial hacking to liberate your devices. You paid for them, after all. You should be able to do whatever you want with them (and you will). Onwards!

3.2 Kernel Ring Buffer

What on earth is a "kernel ring buffer" (KRB)? Ever heard of dmesg(1)? dmesg(1) is what you use to read the KRB. Not so scary now. Is it?

Why is the KRB important? Well: when you plug in (or out) a device, you can see the messages show up in the KRB. If you learn how to pay attention to the KRB, when you are working with hardware, you will become a lot better at trouble-shooting your own problems. Take strings you don't understand and plop them into your favourite search engine; try the apropos(1) command as well.

As we progress with our project, we'll see how to leverage dmesg(1) to our advantage. Learning proper use of dmesg(1) is an essential skill if you want to improve and maintain your computing freedom; dmesg(1) allows you to demystify the inner workings of your computer and get clues on how to fix problems yourself.

3.3 Drivers

Say you plug in your mouse or keyboard into your computer; or even plug them out. The software responsible for translating the physical signals from the mouse or keyboard, to the intermediary physical devices, to the more abstract layers of your operating system (like stuff you see on the screen) is called the kernel; this is the "Linux" part of GNU/Linux.

The kernel is the layer of software that sits between the

physical hardware and the more abstract levels of software that gives you an “operating system”. When you plug in or out your keyboard or mouse, the Kernel has programs which recognise those types of devices and then loads the appropriate software required to use those physical devices; such software are called “device drivers”.

All of the above is a bit vague. Let’s take a look at what this looks like in practice; I’m going to plug out and plug back in my mouse while staring at dmesg(1):

```
# dmesg --human --follow
...
[Feb19 17:26] usb 7-4: USB disconnect, device number 2
[ +25.036175] usb 7-4: new low-speed USB device number
 ↪  4 using ohci-pci
[  +0.193047] usb 7-4: New USB device found,
 ↪  idVendor=0461, idProduct=4d81, bcdDevice= 2.00
[  +0.000006] usb 7-4: New USB device strings: Mfr=0,
 ↪  Product=2, SerialNumber=0
[  +0.000004] usb 7-4: Product: USB Optical Mouse
[  +0.007570] input: USB Optical Mouse as
 ↪  /devices/pci0000:00/0000:00:16.0/usb7/7-4/7-4:1.0/0
 ↪  003:0461:4D81.0005/input/input18
[  +0.000303] hid-generic 0003:0461:4D81.0005:
 ↪  input,hidraw3: USB HID v1.11 Mouse [USB Optical
 ↪  Mouse] on usb-0000:00:16.0-4/input0
```

We’ll briefly analyse this output and introduce a few important tools in the process.

The first thing to note is this string “using ohci-pci”. It’s time to bring in the Linux-specific tool modinfo(8); let’s take a look at what we’re dealing with:

```
$ modinfo ohci_pci
name:           ohci_pci
filename:       (builtin)
softdep:        pre: ehci_pci
license:        GPL
```

```
file:           drivers/usb/host/ohci-pci
description:    OHCI PCI platform driver
```

That output is quite self-explanatory. We see the name of the kernel module; we see that its a builtin kernel module (which means it's compiled into the kernel). "softdep" stands for soft dependency. We see that the licese is GPL. We see the location in the kernel source tree this kernel module resides. And, finally, we see a short description of the kernel module.

I hope, at the point, you've realised that "kernel module" is synonymous with "driver". See? Not that complicated.

So what does this have to do with our USB mouse? Well: when it comes to interfaces, there's usually a few things that sit between your device and the userspace of your operating system. I'll leave it as a research project for you to figure out what "HCI", "OHCI", "EHCI", "PCI", etc. mean.

The next crucial bit of driver information here is the "hid-generic" part; find out what this kernel module does with modinfo(8).

The next thing I want you to do is have a look at the output of the Linux-specific tool lsmod(8); Note the column headers. grep(1) through the lsmod(8) output for the following strings:

- usbhid
- hid_generic
- hid

The "USB HID v1.11 Mouse" from our dmesg(1) output should give us a good idea of what's going on here. Don't know what "USB HID" means? Look it up. Find out what the above kernel modules do, from the stuff you've already learned so far.

Let's take a look at some sample lsmod(8) output:

```
$ cat <(lsmod | head -n 1) <(lsmod | grep hid)
Module                  Size  Used by
```

```
mac_hid                16384  0
hid_generic            16384  0
usbhid                 57344  0
hid                   135168  2 usbhid,hid_generic
```

You've now got a bit of background knowledge to make sense of what's going on when you plug things in and out of your GNU/Linux unit.

3.4 Operating Systems

We're going to be a bit adventurous with our choice of OS to put on the Raspberry Spy. We're going to go with NetBSD; this is a great OS for embedded systems and one you should be familiar with if you plan on doing any embedded work.

NetBSD is an OS with its own kernel and userspace. Thus, NetBSD runs the NetBSD kernel and NetBSD userspace utilities; this is in contrast to the Linux kernel and GNU userspace (GNU/Linux)[1].

NetBSD is quite a beginner-friendly BSD because it has ample documentation; the fact that NetBSD has the primary focus of portability also means you can learn a great deal about portability from several perspectives.

A side note here. Avoid usage of package managers. They are bad for your freedom; to most people, package managers are entirely opaque systems that turns the computer operator into a mere consumer. Learn how to build your software from source code. This way you see all the dependencies[2].

[1] Technically, there's also different bootloaders to worry about but we're going to ignore bootloaders for now as we have enough to deal with. It's also very unfair to GNU to just call it "userspace"; GNU gave the world things like the GNU Compiler Collection and GNU Autotools - things without which much of software today wouldn't exist; there seems to be mass amnesia in the computing world around this, whether it be deliberate or not. And guess what? GNU was about freedom, first and foremost.

[2] i.e., how much junk the software you want to use depends on. It's a great way to filter out bloatware. You will also be able to learn to spot "common denominator" programs software of a certain type depends on. Often, this will enable you to refine your criteria for a program in order to find exactly what you need - opposed to what you think you need (or what others make you think you need).

The opaque package manager is exactly how the Raspberry Spy Foundation smuggled in spyware into the Raspberry Spy. If you build all your programs from source code, you would be less vulnerable to these espionage tactics[3].

You should be the operator of your computer, not a “user”. A “user” is effectively being “used” because they are treated like stupid consumers that get dictated to by other people. Don’t fall for this “user” trap. Be the operator of your computer; take back control; education is the true path to computing freedom.

Note that a lot of these operating systems we’re talking about follow some version of the POSIX specification (with varying degrees of compliance).

3.5 Special Files

It’s important to understand how special files relate to device drivers. What’s a special file? Glad you asked.

Let’s take a look at our friend dmesg(1) as we plug in our UTUB:

```
1 [Feb22 12:13] usb 7-1: new full-speed USB device number
  ↪  3 using ohci-pci
2 [  +0.202882] usb 7-1: New USB device found,
  ↪  idVendor=10c4, idProduct=ea60, bcdDevice= 1.00
3 [  +0.000006] usb 7-1: New USB device strings: Mfr=1,
  ↪  Product=2, SerialNumber=3
4 [  +0.000003] usb 7-1: Product: CP2104 USB to UART
  ↪  Bridge Controller
5 [  +0.000003] usb 7-1: Manufacturer: Silicon Labs
6 [  +0.000003] usb 7-1: SerialNumber: 010C48B4
7 [  +0.024088] usbcore: registered new interface driver
  ↪  usbserial_generic
8 [  +0.000010] usbserial: USB Serial support registered
  ↪  for generic
```

[3] However, don’t think you’re entirely immune, if you compile everything from source. Much has been infiltrated at the source code level.

```
[  +0.003272] usbcore: registered new interface driver
 ↪  cp210x
[  +0.000025] usbserial: USB Serial support registered
 ↪  for cp210x
[  +0.000081] cp210x 7-1:1.0: cp210x converter detected
[  +0.010528] usb 7-1: cp210x converter now attached to
 ↪  ttyUSB0
```

Bit of a mouthful. Let's break it down into pieces that we can actually digest:

- Take a look at the Linux kernel modules usbcore, usbserial, and cp210x with modinfo(8). Not so scary now. Is it?
- Next, have a look at the line "usb 7-1: cp210x converter now attached to ttyUSB0". You should understand all the lines leading up to this one; however, we need to do a bit of digging to find out what this whole "ttyUSB0" business is about. We'll look into some other helpful things in the process.

Here we have a special file called ttyUSB0; So uh where is this file? Let's see:

```
$ find / -name "ttyUSB0" 2> /dev/null
/dev/ttyUSB0
/sys/class/tty/ttyUSB0
/sys/devices/pci0000:00/0000:00:16.0/usb7/7-1/7-1:1.0/t⌋
 ↪  tyUSB0
/sys/devices/pci0000:00/0000:00:16.0/usb7/7-1/7-1:1.0/t⌋
 ↪  tyUSB0/tty/ttyUSB0
/sys/bus/usb-serial/devices/ttyUSB0
/sys/bus/usb-serial/drivers/cp210x/ttyUSB0
```

The path we really want here is "/dev/ttyUSB0"[4]. Time to do a quick check:

[4] The other paths are just as interesting. See Appendix B for details on the specifics.

```
$ ls -al /dev/ttyUSB0
crw-rw---- 1 root dialout 188, 0 Feb 22 12:13
 ↪  /dev/ttyUSB0
```

The “c” in “crw-rw--” tells us that this is a character file. The “188, 0” tells us that the “major” and “minor” number, respectively, of this special “character file”. These files are created with mknod(1). The following can be a useful pointer, when you are lost:

```
$ file --mime /dev/ttyUSB0
/dev/ttyUSB0: inode/chardevice; charset=binary
```

Good stuff. We’re getting somewhere. To find a full list of what these major and minor numbers refer to, we can have a look in the Linux kernel source tree:

```
$ less linux/Documentation/admin-guide/devices.txt
...
 188 char       USB serial converters
                  0 = /dev/ttyUSB0      First USB
                   ↪  serial converter
                  1 = /dev/ttyUSB1      Second USB
                   ↪  serial converter
                    ...
...
```

That’s that part demystified. Isn’t learning great? Now you know where to get the right numbers if you want to use mknod(1) manually on GNU/Linux systems[5].

Now what does all of this mean? We essentially have “cp210x” which is a discrete Linux kernel module; this Linux kernel module is then “attached” to the special file ttyUSB0; it’s this special file ttyUSB0 that the program picocom(1) will be attached to, in order to perform serial communications.

[5] A skill every GNU/Linux operator should have.

You can also see where the different parameters like “idVendor” and “idProduct” come from by taking a look at the appropriate path in the Linux kernel source tree:

```
$ find ./ -regex ".*cp210x.*"
./drivers/usb/serial/cp210x.c
$ less drivers/usb/serial/cp210x.c
...
        { USB_DEVICE(0x10C4, 0xEA60) }, /* Silicon Labs
        ↪  factory default */
...
```

On GNU/Linux systems, you should also take a look at the path /usr/share/misc/usd.ids:

```
$ less /usr/share/misc/usb.ids
...
10c4  Silicon Labs
...
        ea60  CP210x UART Bridge
...
```

Now let’s have a look at what it looks like when we pull out our UTUB:

```
$ dmesg --human --follow
...
[Feb22 15:45] usb 7-1: USB disconnect, device number 3
[  +0.000384] cp210x ttyUSB0: cp210x converter now
 ↪  disconnected from ttyUSB0
[  +0.000164] cp210x 7-1:1.0: device disconnected
```

There you have it! You should understand what’s going on in that output, with your new knowledge of Linux kernel internals. Remember, tools like lsmod(8), modinfo(8), and dmesg(1) are the first things you should look at when you plug things in and out of your GNU/Linux box. This stuff is incredibly

simple, if you know where to look; now you know where to look! No need to be afraid.

Finally, we have the commands:

```
1 $ lscpi -k
```

and

```
1 $ lsusb -t
```

You now know enough to figure out yourself what you get from lspci -k and lsusb -t[6].

You now have a healthy dose of knowledge injected into your grey matter to enable you to handle special files on GNU/Linux systems[7].

[6] Don't know what the options mean? RTFM.

[7] Some of this special file handling knowledge applies to other POSIX-like operating systems as well, with minor details changed.

Chapter 4

Doing The Task

We've now covered enough ground to make the installation of NetBSD on our Raspberry Spy (over our UTUB) a relatively painless matter.

Let's go through the process in little steps.

4.1 Preparing The Boot Media

I'm going to grab the appropriate NetBSD image by taking hints from the following:

- NetBSD/evbarm on Raspberry Pi tells us everything we need to know to pick the right image. All the sections here related to booting are worth reading at least once. Also read sections about consoles and serial consoles at least once.
- Raspberry Pi boot modes is useful if you want to dig deeper into the booting mechanisms of the Raspberry Spy. USB mass storage boot is particularly useful for booting off USB. Trust me, you don't want to muck around with SD cards; they're a nightmare.
- NetBSD/evbarm can be referenced for general information about NetBSD on ARM boards.

The above links should give you a good idea of what's going on and what needs to be done with regards to putting a NetBSD on a boot media that goes into a Raspberry Spy.

Let's go through a concrete example.

My Raspberry Spy is of the model “3 B+” variety so I’m dealing with an ARM64 CPU architecture. We’ll follow along the instructions outlined in Installation procedure for NetBSD/evbarm; pay close attention to the section “NetBSD/evbarm subdirectory structure”; I follow these instructions as I explore Index of pub/NetBSD/NetBSD-9.1/evbarm-aarch64/.

I grab the appropriate image like so:

```
$ mkdir ~/Downloads/netbsd
$ cd ~/Downloads/minted
$ wget https://cdn.netbsd.org/pub/NetBSD/NetBSD-9.1/evb
↪  arm-aarch64/binary/gzimg/arm64.img.gz
```

Now that we’ve got the image, we can write it to our boot media. I’m going to assume you have an appropriate reader already plugged into your GNU/Linux box. I’ve got my USB thumb drive as “/dev/sdg” on my system. Use the right block device file on your system[1]. We base our procedure along the lines of “Installation for ARMv7 and AArch64 devices with U-Boot” section from Installation procedure for NetBSD/evbarm:

```
$ gzip --decompress --keep arm64.img.gz
# dd if=arm64.img of=/dev/sdg bs=1M conv=sync
↪  status=progress
$ lsblk -f | grep sdg
```

We’re going to ignore the minutiae of writing to block devices, bootloaders, and other adjacent topics related to the utilities we just used; that’s left for another time. We care about learning how to use a serial console in this project so we must stay focused on our primary target.

We’re going to have a look at how to make a serial install possible via some editing of the “cmdline.txt” file that now resides in the boot media (on the boot partition which is of type “vfat”):

[1] The command lsblk -f should help you out here. Don’t wipe the wrong device by accident.

```
# mkdir /media/netbsd_image
# mount /dev/sdg1 /media/netbsd_image
# grep "console" < cmdline.txt
# root=ld0a console=fb
# grep "enable_uart" < config.txt
# enable_uart=1
```

The “console=fb” part is to get out OS image to use the HDMI output. We will get rid of that string from the file “cmdline.txt”. Who needs that anyway? One way to do it[2]:

```
# ed cmdline.txt
21
,p
root=ld0a console=fb
1
root=ld0a console=fb
s/console=fb//
,p
root=ld0a
wq
11
# echo ",p" | ed cmdline.txt
11
root=ld0a
```

Remember to check your edits!

We also ensure that “enable_uart=1” is set in the file “config.txt”:

```
# echo ",p" | ed config.txt
82
arm_64bit=1
```

[2] If you use another text editor, that’s fine. You really should learn ed(1) at some point though, especially if you want to get into embedded systems.

```
kernel=netbsd.img
kernel_address=0x200000
enable_uart=1
force_turbo=0
```

Everything looks good! Additional useful information on the Raspberry Spy UART can be found in UART configuration. Pretty self-explanatory. That wasn't so hard. Was it?

Note that the following links document the files we've been messing around with:

- The Kernel Command Line
- config.txt

It's a good idea to back up the state of your image, at this point[3]. We can now safely unmount our boot media and get on with the project:

```
# cd ~
# umount /media/netbsd_image
```

We change directory, before we unmount, so that we don't get any "device busy" errors.

We've now got our boot media ready. Onwards!

4.2 Connecting Physical Components

Before you power up your UTUB, you should really check that the pins are working properly. The very basic test you should do is to check that the right voltage is being supplied. Check out Appendix C.

The pins on our UTUB and Raspberry Spy that we're interested are the following:

[3] At least keep track of the files that you tweaked. If you use some sort of version-control-system, you get bonus points.

- Raspberry Spy: Pin6 (Ground), Pin8 (GPIO14, TXD), Pin10 (GPIO15, RXD). You can find the layout in the official GPIO page.

- UTUB: I've got a CP2104 UTUB so I've got to only worry about the pins marked TX, RX, and GND. I have other pins on the module but they're not relevant for this task.

We won't be using any of the voltage pins on the boards because it's more prone to errors. Just use the USB power supply that comes with your Raspberry Spy.

Don't plug anything into power for the following sequence. Connect the jump-wires like so:

- Ground on UTUB to Ground (Pin6) on Raspberry Spy.

- TX on UTUB to RX (Pin10) on Raspbery Spy.

- RX on UTUB to TX on (Pin8) Raspberry Spy.

Don't make the rookie mistake of matching TX with TX and RX with RX; TX always goes to RX and RX always goes to TX. Keep this in mind, always, when working with UARTs. Colour-coding your jump-wires helps.

We'll just go over the order of attaching the stuff to do with power on our devices:

- Attach the USB power adapter to the Raspberry Pi without plugging the adapter into the power outlet.

- Attach the UTUB to your GNU/Linux box.

- Attach your USB power adapter to your power outlet.

The logic for the above procedure is that you can ensure that your serial interface is up and running before you start getting input from your Raspberry Spy.

4.3 Using Picocom

Using picocom(1) is simple. All we need to do is select the correct baud rate and give the right device file as a parameter to picocom(1).

I'll give you an extract from the manual page to enlighten you:

```
In effect, picocom is not an "emulator" per-se.  It is a
simple program that opens, configures, manages a serial
port (tty device) and its settings, and connects to it
the terminal emulator you are, most likely, already
 ↪  using
(the terminal window application, xterm, rxvt, system
console, etc).

When picocom starts it opens the tty (serial port)
given as its non-option argument.  Unless the
--noinit option is given, it configures the port to
the settings specified by the option-arguments (or
to some default settings), and sets it to "raw"
mode.  If --noinit is given, the initialization and
configuration is skipped; the port is just opened.
Following this, if standard input is a tty, picocom
sets the tty to raw mode.  Then it goes in a loop
where it listens for input from stdin, or from the
serial port.  Input from the serial port is copied
to the standard output while input from the standard
input is copied to the serial port.  Picocom also
scans its input stream for a user-specified control
character, called the escape character (being by
default C-a).  If the escape character is seen, then
instead of sending it to the serial-device, the
program enters "command mode" and waits for the next
character (which is called the "function
character").  Depending on the value of the function
character, picocom performs one of the operations
described in the COMMANDS section below.
```

We use “C-a C-x” (Ctrl+a followed by Ctrl+x)[4] to tell picocom(1) to exit; for more, RTFM; in particular, pay close attention to the “COMMANDS” section.

Make sure you’ve set up all the physical connections, as advised. It’s time to attach our UTUB to our GNU/Linux box and then make sure we invoke picocom(1) correctly:

```
# picocom --baud 115200 /dev/ttyUSB0
picocom v3.1

port is        : /dev/ttyUSB0
flowcontrol    : none
baudrate is    : 115200
parity is      : none
databits are   : 8
stopbits are   : 1
escape is      : C-a
local echo is  : no
noinit is      : no
noreset is     : no
hangup is      : no
nolock is      : no
send_cmd is    : sz -vv
receive_cmd is : rz -vv -E
imap is        :
omap is        :
emap is        : crcrlf,delbs,
logfile is     : none
initstring     : none
exit_after is  : not set
exit is        : no

Type [C-a] [C-h] to see available commands
Terminal ready
```

[4] I don’t know why the manual doesn’t bother to explicitly mention that these are GNU-Emacs-style key sequences.

It really is that simple. You've now got a serial terminal ready and listening.

4.4 OS Installation

Now that you've got a serial terminal operational, all we have to do to install NetBSD on the Raspberry Spy is to plug the USB power adapter into the power outlet. Keep a close eye on what goes on in the output of your serial terminal:

```
...
[   7.4246937] root device:
[  11.6252523] use one of: mue0 sd0[a-p] ddb halt reboot
[  11.6252523] root device: sd0
[  13.9755661] dump device (default sd0b):
[  15.7257992] file system (default generic):
...
```

You should be promted to pick a root device. I pick "sd0" as it's the first 'disk' offered by NetBSD (which can only be my boot media)[5]. I go for the suggested defaults, for everything else. No need to overcomplicate things, at this point.

You will probably see your Raspberry Spy reboot once or twice during the OS install process. Just pass the same parameters for the boot device, and you should be good to go.

Eventually, you should be met with the following:

```
...

NetBSD/evbarm (arm64) (constty)

...
```

[5] See the NetBSD sd(4) manpage for details.

```
6
7 login:
```

If you login as “root”, you should have a nice login shell presented to you.

And we are done! You’ve successfully done some tinkering over a serial terminal. That wasn’t so hard. Was it? You can shutdown your device (halt the OS) like so:

```
1 # shutdown -p now
2 ...
3 [ 910.5814809] The operating system has halted.
4 [ 910.5814809] Please press any key to reboot.
```

You can now disconnect the power supply from your Raspberry Spy. Then just send “C-a C-x” to picocom(1); after which, you should see:

```
1 ...
2 Terminating...
3 Thanks for using picocom
4 #
```

Welcome to the world of serial terminals; hack your heart out!

Chapter 5

Thanks

We'd like to take the opportunity to thank you, the reader. We believe everyone deserves a computing education; however, the topics of computing freedom and how computing affects our basic human rights are neglected in computing education today; at E2EOPS PRESS we strive to change this. Our goal is to inform, educate, and inspire. Computing is also a lot of fun! We want everyone to experience the joys of computing. We hope you enjoyed this issue of our periodical as much as we enjoyed bringing it to you!

Our work requires research, equipment, and infrastructure to deliver. We strive for the best quality in all we do. If you would like to support us, there are several ways you can do so. Any support we get from you enables us to bring you the best we possibly can.

We distribute all our periodicals via peer-to-peer technology. There are things we publish that some people don't want out in the open. Thus, if you can contribute to the peer-to-peer sharing, you would be helping us out immensely!

If you would like to support us by making a cash donation, we have a Paypal account that you can send donations to:

- https://www.paypal.com/donate?hosted_button_id=B5VPZJBKLL2S6

For those that like to use QR codes, you can use the following QR code to donate to our Paypal:

If you'd like to donate in some other way, you can send an email to

donations@e2eops.io

and have a chat with us about it.

For encrypted communications, you can use the OpenPGP Key provided in chapter 6.

And, as always, happy hacking!

Chapter 6

OpenPGP Key

At E2EOPS PRESS, we take your privacy seriously. If you want to send us an encrypted message, you can do so with the following OpenPGP key:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQGNBF7JMNYBDACeCfa+NpFSOUOyIKqdZdtIjEZCaLzBKQmnqLJo/5BZZiCrGK8+
MYUfBz6iiEqEJf7N0R6Kg/esyIEy35uYdiLr66fg/sEXHk5eW/9Hanex/Igk9nmg
1+eGo1sk/MQh7lUNydqOHVokhjcPHleUGiJE/F0MjDvs/xTN1HYf1uhmJOZbpL/D
oTcTssL1BB2b95QfuUViX7I/+Xoe4VSaxdVlVGGCSI8jAPqb+0PYX6N2yHllFaRI
8D4cQkKpP5z2Y9m65ALWsR0M6GBUIeYe4QzfBVwuPg7TAiSpZWDC3vPIWFsC1zbL
NT4y6bxjv9gE0S3GVXZC/bTHuVL3FNrStnMFX0z8E4c19vyo9TyTxXLKC7Qy/dWJ
Nf+3tbgNQ7yR8MBwciyQho9MltrIFTz+JXZj7xCM0QiYgTo4rBl9eczaGLZwlOJn
sybs3Ia0ZTfm5yQAQ3eJ4yKKkiCjADF6fq6AVgV0D36wgBiZnVa+zbSteamUM7L8
JN4rzqfm587d+/0AEQEAAbQgR2F2aW4gTC4gUmViZWlybyA8Z2xyQGUyZW9wcy5p
bz6JAdEEEwEKADsCGwMCHgECF4AFCwkIBwIGFQoJCAsCBBYCAwEWIQTHhRt/mdsE
uRCTt6QUP5lU/9Mq/wUCXyPcLAIZAQAKCRAUP5lU/9Mq/xtpC/9nEcxWrB5e3W1o
Xl8W9uumnwG5o2ZUYfHtQ+QVpkowdQgEFPLTBicWRPnb61eaT/60hkhH68Gb1oVZ
BHHN3YNF9qqQHqOT1iAuP0nidkHXUaz4Bbl0wR8brvY2aJO4l03sOWztQHkAoo+u
wNohHM1lwqKYN+HRWm+JyTRxw/YHkYI1Rz1aN9hUNTax1HqAm5fSq+R/VKyUZ9tI
2D93p2PK+c1SEJS1goBnlJu4HW2T7NA7oUboKG0J4h2yBV6WtSgskFEWqYEBzvxw
ZaWzooOnJRPc+CibhRuH9VJvegRCkwZcNuZfHBKkh+d30LaJJlpifnu9ADs0PkXx
HUsHlgMjUbzjztFEBmAjS8PvKOG2nJxcv362749/M6vI7KP7ktf4i+3wZLe8BC5y
hSAAQtLUBC/IratA+4NSI8lxDlNB+rrFTXzYTVGscJWlACCK4l0xaXNC0OBcRosH
oFMgSogrXrQ2SsSu8oeVOjx/HOd8tThrCKd2nd40iG5Z3fftzP+5AY0EXskw1gEM
ANpbLPGSE9gjT8aAkcK0gIhNwTwR/X9T1YCKViiOYALCzPfGYMQqATSe/J7o/ysO
+lAB0B3NExdIbwh3knhoEsXdx9zNz18q827aCdLJVuq9kizXoCkWchJ0QjgacvNe
0jGZ+o3ULIMmnWztGzNixZaAiwzalWghFuiX4pG0su8TYAN1T6DGvdb+F1rx0xo8
tb3T8RaO0VZyji3zBBNXD1ECU6dmi0AJ0zD0S5iRIXIn+QfRR2oJu9ZSR6En7PS1
b4bhJ2OnBE4ZQ3mS4aHPaeIWbeCeq8EuaVQNyv5jT3wD18DIRigOxl9XJFy1IDO+
A45hz9FwxULH1QH5bZXRmkYfcxkLMqoMci0+8XxobpmILjkSHeAo4PR8KTHpNUXf
I7rJLpufYRTHf5votrJmgbea5bfnAlSoOXMlN2XZ9oJimOTfyzItCSwb0pkghnW0
```

```
Hyo3kd5RbDcZrd6RSSVWo/41gr8SXatlkpbnEYjktFGe2Nxls6qZAgA/lqKrS9ti
7wARAQABiQG8BBgBCgAmFiEEx4Ubf5nbBLkQk7ekFD+ZVP/TKv8FAl7JMNYCGwwF
CQPCZwAACgkQFD+ZVP/TKv9XAwv9Heor5YuZalOcXXiGCrqYNHuQJLbDvUpTK6wI
t9peV4t6E8oiqb0/0ezLGQSlP8RkBKUkhslVn5M6cDIxdCAlZxUIOjF1tJyHrTua
wEayHO5q84tixa00+x2/mmaDy0ddZXc4gITLgZjN4J3e9GUww4N3TUVJTZDPF+mP
eFd2HlGj3AHLtNEfsEw4HvYNNzexnom66gdDrrCIBxWRgUD2aRQKxQOquFQt2zYS
dgOXVS0eyaN81uqFHVh+tvyBKkrK8D/recaPcCXAFSv9UUcQA6jEFdC9WEjgHD7J
nRrCwOGBTZ7IvYO5+rpyh/w8k6eZYhUm3UdXYuPsxaCsSL+OW46DBP0C5sKrhxZK
thZOhll4ZYrfX1Vv4awBAB6iMZ0i+i39AQPmpvUviFGUfLkHEEsarHgPDchaOv1p
0xa11jUxU5nkqfiQj5A4h4TMP0q9tGdf1puxRZqwp8T0Jc9uQAm6wUMaaaugiXn+
BQfF5XIfOiXZx9DThhqYwO4WztdEuQINBF87rdYBEAC66OFgJa3p9d6Ets5A3NVm
XxXkCBmgIg1HG44bmKz3o2xudRP6o3bbmVI06cmfGAVmKJ0voAVsgHMsTg30iZb3
D/X9iRYTB/1suyYIgP9cRxltLLMTL4LycmrCRCnTJSMThA1V2d5brmmrWcOPmqnT
CdbOjUD5OWzGsfIkP0t+Ef4v94j574WAvpGYd7Wf5o30JKNfdOjdLOVFdp6lV9mM
FxebC1lZ6uRSeL77/XZJC4ZQ8TzhgsdgnE4M1sKvaDO+DTS3pu7+Dh+ORFQ9TQDP
ZuMBpln3UBa1evVR8bF9SgqjGo2Al8K590xGNJEHzj+BGqEKjOVN05uMoplByMdZ
pD2lRMdqgIqVXiNtBSBhv1oGQhu5fxW11Xv4Kh5BzuOMLS/0KT2cdlk06eCgWZqT
/IM5ipKYpJJgtkJ/0sU+hLq+jilAr3ZhmkxCim+9uYOZ93ypy7QiW8ldbsGEDh5S
KcU+qWGNEMHhNwuHY9ChOlhcWi/q5/mZTKXG60/q6RZRI2QxbuKsX3c8GbEea2jU
Q+v4zFp/x7G1aPRTRtqOssvBtIikBiKWBEOBwYwqpEjD6IfmWVh8wlJZx/tcBRGF
0FGpyieGAFINaQIjKURm7bHkn4bQwIZkmbAssA6ZHcl9u6/u60155K4ifuf7ChyK
8H5n6vJq4AUxQkA8kADvnwARAQABiQG8BBgBCgAmFiEEx4Ubf5nbBLkQk7ekFD+Z
VP/TKv8FAl87rdYCGyAFCQHhM4AACgkQFD+ZVP/TKv9Fwgv+P20Qu7USOJfaUVN4
I52GS6R8veww4aKY16ts3XlIUl3mqbH3KoB1T+YKuqoVrNCtGeXVD0AB+BJxxrQS
BFtYq4ZgPnSo7NQST6SWbWhzJmpfuhoInGb9l8v0mwD50DrOPjeZcX7TrQN4jWft
OJQkKviCYFCu36xa84FUYxTyvA5V4TpWAgPTUVBpf8BrI2Ktw3Wc1THHqcwIszec
Mr/mexDfdCa8G78u3rgX4kZPSGFwaGK339eqlb7rMtspkKQPPmm92lYdfV/icbvw
7iaMAirgYq72qzkuIV6GMUKsTa/1wkJ54yACaLXS9A3NXDVizErWHam9TG/ce/Ll
EPUsrXTesrhqZpaHHNxnfiQOXzocrldvvdYOxBeGfMQY7fftX8GgQwMjMi2kracr
hWc1mCSMSCerTWCtRfw0CK4Au6881rAAmXDwvjU586ezv1MdBqVmN7eOeeVVGdM1
/+S5QP//oK6MEkAS8y2ZP8A/3iMlYz+SBOUSD0AJ0RZEhkhr
=JMTx
-----END PGP PUBLIC KEY BLOCK-----
```

Appendix A

Malicious Hardware

While doing research for this issue, I often ran into USB-to-UART bridges of the “FTDI” variety. Upon further digging, an ugly bit of history surfaced. The FTDI modules have a reputation for sabotaging people’s hardware.

Sadly, we live in a world where this sort of thing is the norm. Pay close attention to the products you buy. You need to practice vigilance in order to defend your computing freedom. Remember, you have control over your wallet. Don’t support malicious actors, if you have the choice (in this case you almost certainly do).

Appendix B

Linux Kernel Source Tree Analysis

The directory trees rooted at /sys and /proc are mapping of Linux kernel data structures and interfaces; you can read up on these in the Linux kernel source tree from:

- linux/Documentation/filesystems/sysfs.rst
- linux/Documentation/filesystems/proc.rst

You don't have a local, up-to-date, copy of the Linux kernel source tree? You really should. Note that some of this documentation is hilariously out-of-date; use the git log on a file to see the last time parts of a file was given an update:

```
1 $ git log -p filename
```

This should give you what you need. Since the Linux kernel is developed with Git, it pays dividends to learn at least the fundamentals of Git.

It's a frequent occurence that people ask me how to make sense of the Linux kernel. You need the following prerequisites:

- A familiarity with the C programming language. The syntax is easy to pick up for most people because a lot of the popular programming languages in use today are based

on C. Most operating systems today are written in C; the same goes for embedded systems. If you don't have a good grasp of C, you can kiss any hopes on working on this stuff goodbye. C is not as hard as people make it out to be; just look at real code and don't waste your time on pointless exercises. Start with the smallest real-world programs you can find - like echo(1); once you get the simple stuff, get more ambitious and look at more complicated things. The following resource is also invaluable to the novice C programmer: C reference.

- To make sense of other people's C code (particularly spaghetti), you need a good source code tagging system. I recommend GNU Global because it works well on most Bourne Shells. Using GNU Global will enable you to look up definitions for things like functions and structs in C code easily.

- You need to learn GNU Autotools to automate the workflow of building makefiles and such. The old "./configure && make && make install" ritual stems from GNU Autotools. Learn it and embrace it. You can build truly portable software once you learn the fundamentals of GNU Autotools. You won't understand head nor tail of embedded programming with the Linux kernel (and several other things) unless you have a grasp on the rudiments of GNU Autotools.

- Whether you like it or not, Git is an essential part of Linux kernel development. Without a firm grasp of Git fundamentals, you won't get anywhere. While you're at it, you should look into the standalone utilities GNU diff and GNU patch; Git is essentially an abstraction on top of these tools.

You should now have enough pointers to begin acquiring knowledge about how to make sense of the Linux kernel (and a whole lot of other things). The aforementioned prerequisites abstract to OS and embedded development and being an effective operator of your computer. These are the tools you really need to know to get anywhere.

All of this stuff applies to several other things. Once you start learning them, you'll see what I mean. It really isn't a lot to take in. Knowledge of this stuff will last you a lifetime. Don't fall for the IDE X or framework Y bullshit; those are moving targets and are deliberately broken

to keep people reliant on the dictators for “support”. Educate yourself; it’s the only path to computing freedom. Become an operator; don’t be a mindless consumer.

Appendix C

Digital Multimeter Tests

As always, follow the instructions in the manual of your Digital Multimeter (DMM). RTFM extra carefully, otherwise you end up with magic smoke (why you were recommended spares).

There really are only two simple things you need to test on your UTUB:

- Voltage coming out of the UTUB TX and RX pins.
- Current from the TX and RX pins.

There's not really much more to be said here. The one bit of general advice is to use a breadboard and some jump wires, if you have access to one; crocodile clip test leads for your DMM also make life easier. Basically, try making sure you don't short circuit your UTUB by having DMM test leads too close to each other.

Make sure the test leads are plugged into the appropriate terminals of your DMM. Always make sure the fuse of a DMM terminal is sufficient for what you're measuring.

You can find GPIO voltage specifications of the Raspberry Spy in the official GPIO guide. Make sure you cross-check with the right CPU model's datasheet.

You may end up needing to buy some resistors to get the right voltage and current. You can find background infor-

mation useful to the novice hardware hacker from the excellent Sparkfun tutorial on pull-up resistors; follow the appropriate links to fill out gaps in your knowledge. However, most UTUBs are usable out-of-the-box (OOTB) so you shouldn't really have much issue here. But it doesn't hurt (unless you zap yourself) to get a bit of electronics background knowledge since you're playing around with wires and electricity!

Index