



Jan-Paul-Herr, pixello.de

Un script Perl que controla un cañon USB de juguete

# JUEGOS DE GUERRA

Aunque un juguete USB como el lanzador de cohetes de poliestireno sólo incluye un CD para Windows, funciona sin problemas bajo Linux con un poco de ingeniería inversa. Gracias a libusb, esto no requiere ni compilar un driver de dispositivo: Perl controla el dispositivo desde el espacio de usuario.

POR MICHAEL SCHILLI

Problemas en la oficina? No necesariamente tienen que conducir a una batalla, como la del vídeo “The Great Office War” [2] del fabricante de juguetes Hasbro. Incluso si no tenemos un plan de ataque, el lanzacohetes controlado por USB Rocket Baby (véase la Figura 1), del fabricante chino Dream Cheeky, es una auténtica ganga por menos de 14 Euros o de 20 dólares. No sólo levantó los ánimos de mis compañeros de trabajo, sino que también me dio la oportunidad de estudiar el complejo subsistema USB [3] del núcleo de Linux.



Figura 1: El lanzacohetes USB Rocket Baby, de Cheeky Dream.

Al abrir la caja encontramos un CD para Windows XP, pero ni rastro de un controlador para Linux. Esto parece haber provocado que un buen número de fanáticos de los gadgets en la comunidad de desarrolladores hayan investigado el protocolo USB que usa el juguete en Windows con sniffers de puerto USB, tales como USBsniff, para hacer la ingeniería inversa de la interfaz y crear bindings para varios lenguajes como Python, o incluso sistemas operativos completamente diferentes [4].

Cuando el juguete se enchufa a un conector USB, la distribución Ubuntu Hardy Heron lo detecta automáticamente. Los mensajes del kernel, que pueden leerse en el archivo de log `/var/`

`log/messages` (véase la Figura 2), nos indican que el lanzador de cohetes está conectado al controlador UHCI del ordenador basado en Intel.

De acuerdo con el archivo de log, el subsistema USB del kernel ha mapeado el lanzador de cohetes en `usb 5-1`. El árbol de directorios `sysfs` bajo `/sys/bus/usb/devices/5-1` nos da los detalles. El sistema de ficheros USB, `usbfs`, proyecta los datos USB internos del kernel al espacio de usuario de esta manera. La Figura 3 muestra que el valor `idVendor` del lanzador es `0x0a81`, y el de `idProduct` es `0x0701`, como puede determinarse con facilidad mirando el contenido de los respectivos archivos en el árbol de directorios de `sysfs`. Según el orden en el que se conectan los dispositi-

```
mschilli@mybox:~/mnt/data/big1/mschilli.do.not.delete/DEV/articles/rocket/eg
Mar 1 10:23:06 mybox kernel: [404953.755026] usb 5-1: new low speed USB device
using uhci_hcd and address 4
Mar 1 10:23:06 mybox kernel: [404953.987072] usb 5-1: configuration #1 chosen f
rom 1 choice
Mar 1 10:23:06 mybox kernel: [404954.017977] hiddev9/hidraw5: USB HID v1.00 Dev
ice [Rocket Baby Rocket Baby] on usb-0000:00:1d.1-1
```

Figura 2: Tras conectar el lanzacohetes, el kernel detecta el dispositivo y le asigna una entrada USB.

vos USB, el kernel les asigna un número USB variable: en lugar de *usb 5-1*, la próxima vez podría ser *usb 3-1*.

Dicho esto, sólo hay un dispositivo conectado al PC con los valores *idVendor* e *idProduct* que acabamos de descubrir, por lo que un programa puede encontrar la dirección USB del dispositivo con bastante rapidez y fiabilidad analizando el árbol USB hasta encontrar la combinación correcta.

Linux usa normalmente controladores de dispositivos del kernel para comunicarse con dispositivos USB. Un driver es difícil de programar porque no hay red de seguridad, como se da por hecho en el espacio de usuario. El menor error de puntero puede torpedear el sistema Linux entero y forzar un reinicio. Además de esto, los usuarios tendrían que recompilar el controlador de dispositivo para cada nuevo kernel y cargar el módulo como root ejecutando la orden *modprobe*. Las estructuras de datos suelen cambiar rápidamente en el núcleo, y es posible que el código fuente escrito para el núcleo 2.6.22 haya que tirarlo por la ventana con la versión 2.6.24.

Pero si usted no necesita una alta tasa de transferencia de datos o respuestas en tiempo real, no hay necesidad de lidiar con el kernel para la lógica de control. En cambio, el kernel dispone del sistema *usbfs* que permite comunicarse con los dispositivos USB a nivel de hardware, lo que significa que podemos implementar el controlador en espacio de usuario.

El proyecto de software libre *libusb* [5] proporciona una práctica librería para

programas en C, y el módulo Perl `Device::USB` de CPAN engloba funciones Perl sobre aquél.

La Tabla 2 muestra cómo levantar los cañones del lanzador de cohetes cerca de media pulgada con sólo un par de líneas de código Perl. En primer lugar, la función *find\_device()* utiliza el módulo `device::USB` para localizar el dispositivo con los valores *idVendor* e *idProduct* descubiertos previamente en el árbol de USB. Si tiene éxito, el método *open()* abre una conexión con el dispositivo.

El subsistema USB del kernel soporta cuatro modos de comunicación diferentes para los controladores USB: *Control Transfers* para mensajes cortos, *Bulk Transfers* para grandes volúmenes de datos, *Interrupt Transfers* para transferencias críticas de datos, e *Isosynchronous Transfers* para transferencias de datos en tiempo real. Gracias a la ingeniería inversa, los desarrolladores descubrieron que el lanzador de cohetes utiliza mensajes de control de un byte para mover la torre y disparar los cohetes de poliestireno. La Tabla 1 muestra los códigos de las distintas acciones.

Un código mueve el lanzador hasta que otro código cambie la dirección o una orden de parada cancele el movimiento, lo cual es importante, porque si un programa inicia un movimiento y luego falla, el motor del lanzador de cohetes seguirá funcionando. Los valores hexadecimales pasados al método *control\_msg()* en las líneas 12 y 18 definen la forma en la que la interfaz USB pasa el byte de control al controlador: *0x21* representa el tipo de petición, *0x09* para *USB\_REQ\_SET\_CONFIGURATION*, *0x02*

para *USB\_RECIP\_ENDPOINT* y el valor *0* para un índice no utilizado. Luego viene el código de control (*0x02* para mover los cañones en la línea 12) para manejar el lanzador. La función *chr()* de Perl transforma un valor entero, como el *0x02*, en un único byte que contiene el mismo valor.

Los dos últimos parámetros especifican la longitud de la cadena, *1*, o exactamente un byte en nuestro caso, y el tiempo de espera en milisegundos (1000) antes de que expire el programa.

Tras esto, el programa de prueba realiza una breve pausa de una décima de segundo (10.000 microsegundos) gracias al módulo `Time::HiRes` de CPAN y su función *usleep()*, antes de pasar a enviar el byte de control *0x20*, que el extremo receptor interpreta como una orden de parada, apagando de este modo el motor de la torre del cohete. En la Tabla 2, las dos llamadas a *control\_msg()* mueven así la torre del lanzador durante una décima de segundo. Si la torre no está en su máxima elevación ya, significará que oiremos el motor durante una fracción de segundo y los cohetes de poliestireno se elevarán alrededor de 20 grados.

## ¡Fuego!

Cuando dispare los cohetes, tenga en cuenta que el motor necesita bombear aire durante unos dos segundos para crear la presión necesaria para disparar los proyectiles. Para que el programa pueda saber cuándo apagar el motor tras haber lanzado el cohete, debe acceder a la interfaz USB y leer los datos del controlador del cohete lanzador. El controla-

**Tabla 1: Códigos de Control**

abajo	0x01,
arriba	0x02,
izquierda	0x04,
derecha	0x08,
fuego	0x10,
parar	0x20,
iniciar	0x40,

**Tabla 2: Parámetros para control\_msg()**

<code>\$requesttype</code>	=> 0x21
<code>\$request</code>	=> 0x09
<code>\$value</code>	=> 0x02
<code>\$index</code>	=> 0
<code>\$bytes</code>	=> chr(...)
<code>\$size</code>	=> 1
<code>\$timeout</code>	=> 1000

**Listado 1: rocket-test**

```
01 #!/usr/local/bin/perl -w
02 use strict;
03
04 use Time::HiRes qw(usleep);
05 use Device::USB;
06 my $usb = Device::USB->new;
07 my $dev =
    $usb->find_device(0xA81,
    0x701);
08 $dev->open;
09
10 # Move Up
11 my $val = 0x02;
12 $dev->control_msg(0x21, 0x09,
    0x02, 0, chr($val), 1, 1000);
13
14 usleep(150_000);
15
16 # Stop
17 $val = 0x20;
18 $dev->control_msg(0x21, 0x09,
    0x02, 0, chr($val), 1, 1000);
19
20
21 # Read status
22 $val = 0x40;
23 my $buf;
24 $dev->control_msg(0x21, 0x09,
    0x02, 0, chr($val), 1, 1000);
25 $dev->bulk_read(1, $buf = "",
    1, 1000);
26 printf "Status %08b\n",
    ord($buf);
```

dor reporta qué acciones están disponibles y cuáles no. Si se gira la torre tan a la derecha como sea posible, el controlador devuelve una cadena de estado con un valor de `0x08` (binario `0000_1000`) para mostrar que todas las acciones aparte de `0x08` están ya disponibles. Como se ve en el Tabla 1, `0x08` representa la dirección *derecha*. Si se gira la torre tan a la izquierda y abajo como sea posible, el controlador devuelve un mensaje de estado `0x05` (binario `0000_0101`), porque ambos `0x01` (*abajo*) y `0x04` (*izquierda*) están bloqueados ahora. De manera similar, el dispositivo USB fija una bandera `0x10` (binario `0001_0000`) poco después de disparar para indicarle al controlador que ahora puede mandar un `0x20` para apagar el motor, a menos que quiera disparar el siguiente del total de tres cohetes.

Para comprobar el estado de lanzamiento, el controlador envía en primer lugar un código de control `0x40` a través de `control_msg()` al dispositivo USB, al cual sigue inmediatamente una transferencia en bruto con `bulk_read()` para recoger la cadena de datos devuelta por el dispositivo. La Línea 26 del Listado 1

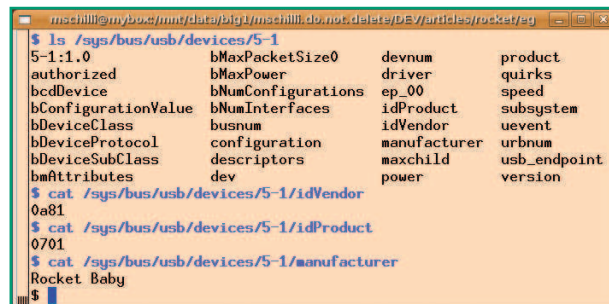
escribe el resultado, que es `00000000` en la mayoría de los casos, a menos que la torre se gire a uno de los límites, o a una elevación mínima o máxima, o justo se haya lanzado un cohete.

El módulo

`Device::USB::MissileLauncher::RocketBaby` de CPAN proporciona

una clara interfaz de abstracción. Un objeto recién construido ofrece los métodos `do()` y `cando()`, que aguardan acciones como las cadenas `left`, `up`, `fire` o `stop`. El método `do()` ejecuta estas acciones, mientras que `cando()` sólo emite una petición de estado y una lectura en bruto para comprobar qué acción está disponible en ese momento.

El Listado 2 muestra cómo usarlo. Para empezar, rota la torre hacia abajo y a la izquierda hasta el límite para determinar la posición precisa, entonces mide el tiempo necesario para elevar la torre completamente y girar completamente a la derecha. Luego divide estos dos tiem-



```

$ ls /sys/bus/usb/devices/5-1
5-1:1.0          bMaxPacketSize0      devnum              product
authorized      bMaxPower            driver              quirks
bcdDevice        bNumConfigurations   ep_00              speed
bConfigurationValue bNumInterfaces       idProduct           subsystem
bDeviceClass     busnum               idVendor           uevent
bDeviceProtocol  configuration         manufacturer        urbnum
bDeviceSubClass  descriptors           maxchild           usb_endpoint
bmAttributes     dev                  power              version

$ cat /sys/bus/usb/devices/5-1/idVendor
0a81
$ cat /sys/bus/usb/devices/5-1/idProduct
0701
$ cat /sys/bus/usb/devices/5-1/manufacturer
RocketBaby

```

Figura 3: Linux muestra los detalles de los dispositivos conectados en el árbol de directorios `/sys`.

pos a la mitad y centra la torre con los valores calculados antes de disparar los tres cohetes, uno tras otro.

## Instalación

Linux necesita el paquete `libusb-dev` en el espacio de usuario para acceder a los dispositivos USB. Cualquier distribución reciente lo tendrá. Es mejor instalar los módulos `Device::USB` y `Device::USB::MissileLauncher::RocketBaby` usando una shell de CPAN. Los diferentes tipos de lanzacohetes utilizan combinaciones de códigos diferentes. En caso de duda, se puede buscar la combinación adecuada en Internet y luego agruparlas en una abstracción como el módulo `RocketBaby` de CPAN.

Existe un vídeo en Youtube [6] que muestra cómo responde el lanzador de cohetes al script `center-fire`. Y ahora un mensaje del Departamento de Seguridad Nacional: por favor eviten exportar este script a países no confiables. ■

## Listado 2: center-fire

```

01 #!/usr/local/bin/perl -w           [gettimeofday] );
02 use strict;                        25
03                                     26 do_until("left",
04 use                                  $right_elapsed/2);
05                                     27 do_until("down",
   Device::USB::MissileLauncher:      $up_elapsed/2);
   :RocketBaby;                       28
06 use Time::HiRes qw(usleep         29 for(1..3) {
   gettimeofday                         30   do_until("fire");
07   tv_interval);                     31   usleep(100_000);
08                                     32 }
09 my $rb =                             33
10                                     34 #####
   Device::USB::MissileLauncher:      35 sub do_until {
   :RocketBaby                         36 #####
11   ->new();                            @_;
12                                     37 my($what, $max_time) =
13 do_until("left");                    38
14 do_until("down");                   39 my $start =
15                                     [gettimeofday];
16 my $right_start =                   40
   [gettimeofday];                     while($rb->cando( $what
17 do_until("right");                  )) {
18 my $right_elapsed =                 41   $rb->do( $what );
   tv_interval(                         42   usleep(100_000);
19   $right_start,                       43   last if defined
   [gettimeofday] );                   44   $max_time and
20                                     45   tv_interval($start,
21 my $up_start =                       [gettimeofday] ) > $max_time;
   [gettimeofday];                     46 }
22 do_until("up");                      47 }
23 my $up_elapsed = tv_interval(        48 $rb->do("stop");
24   $up_start,                          49 }

```

## RECURSOS

- [1] Listados de este artículo: <http://www.linux-magazine.es/Magazine/Downloads/55>
- [2] The Great Office War: <http://www.youtube.com/watch?v=pVKnF26qFFM>
- [3] "Essential Linux Device Drivers", por Sreekrishnan Venkateswaran, Prentice Hall, año 2007.
- [4] Python para Interfaz de un Lanzador de Misiles USB, Pedram Amini: <http://dvlabs.tippingpoint.com/blog/2009/02/12/python-interfacing-a-usb-missile-launcher>
- [5] Proyecto libusb: <http://libusb.sourceforge.net>
- [6] Vídeo en Youtube: <http://www.youtube.com/watch?v=6qTRhDijJc>