Go Go Gadget Python

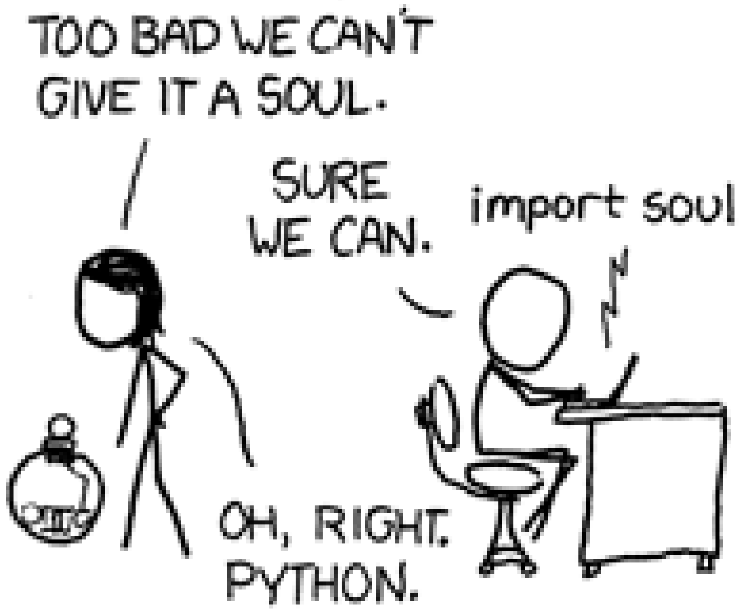
Nick Waite Furkan Cayçı

Hardware for software people

- Gadgets are cool
- Writing drivers is not so easy
  - Usually done in C
  - Requires knowledge of low-level interfaces
  - Can easily crash your box
- Many devices do not need compiled drivers
  - Low data rates
  - libUSB and other abstraction layers
- Is there a way to “rapid prototype” drivers?
  - Fast, easy, fun
  - Cross-platform would be nice

Python

- A very handy scripting language
- Modules for almost everything
- Even hardware…
  - Pyserial
  - Pyparallel
  - PyUSB
- Looks like a winner!



Today's Menu

USB

Serial

The Serial Port

- Electrical
  - Full-duplex
  - Hardware flow control (often not used)
  - [0] + [n]*n_bits + [1]
  - 0 = -3 to -15 volts, 1= 3 to 15 volts
- Mechanical
  - DB-25, DE-9
- Often a USB device pretending to be serial port

Handshaking and cables

- Will add graphics later
  - Much of the confusion in serial land revolves around flow control and what kind to use
    - True hardware flow control
      - Fake local loopback flow control
    - Software flow control (XON/XOFF)
    - No flow control (most common nowadays)
  - DCE/DTE – which side are you?
    - Null modem cables

Serial in python

- It’s easy (mostly)

```
>>> import serial
>>> s = serial.Serial('/dev/ttyS1', 9600)
>>> s.write("hello")
>>> print s.readline()
>>> s.close()
```

- There are some gotchas, however

On actually using pySerial

- There are subtle issues with pyserial's use in robust driver code
  - Timeouts
  - Flow control
  - Buffering
  - Alternating reads & writes
    - Flush ports!

- TO BE COMPLETED LATER

Actual gadgets

- Demo showing actual code
- Demo sniffing serial transactions with special cable?

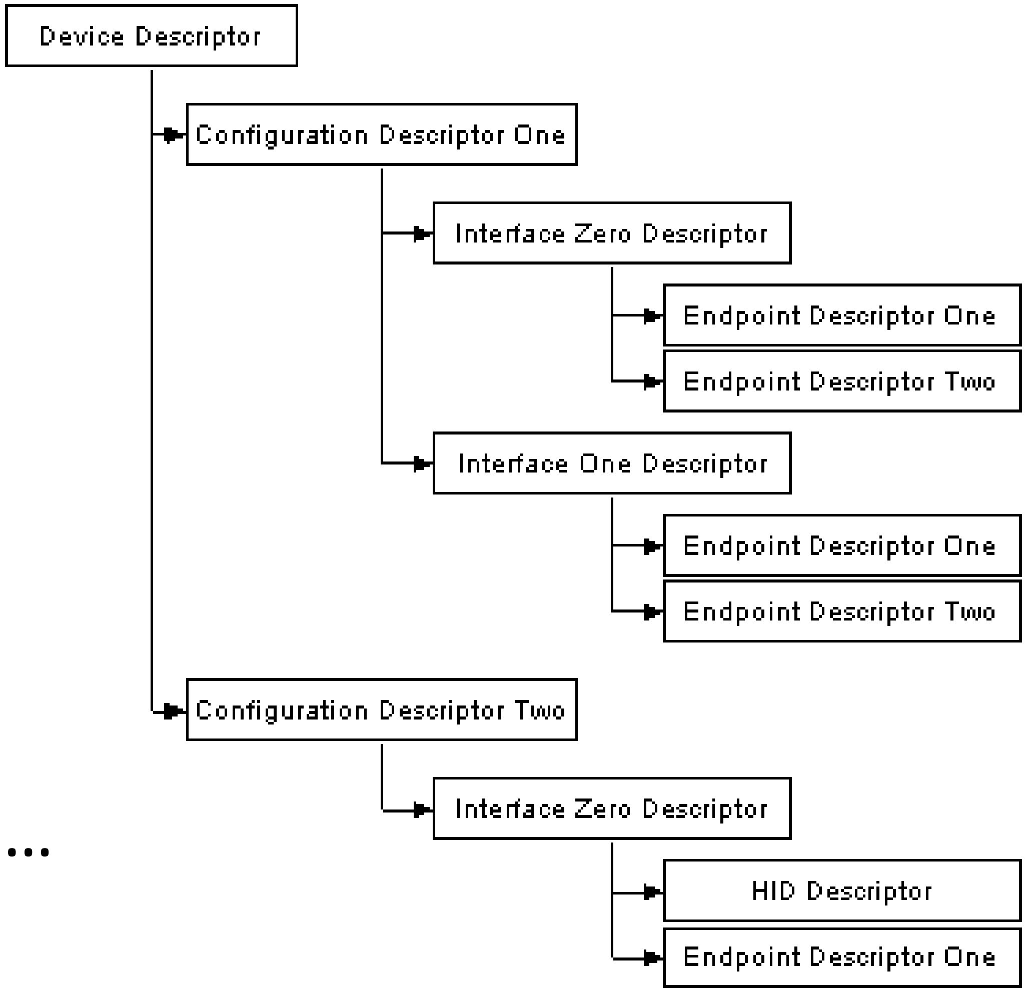
USB

- A great example of a forward looking standard
  - Since 1996, still backward compatible!
- Really is the universal bus
  - Ever seen a PS/2 fondue pot?
  - mmmm….cheese

- Practically, most cool gadgets you will want to reverse-engineer are USB
  - Many will be HID-class

USB made complicated

- A Device has
  - Configuration(s)
    which have
    - Interface(s),
      which have
      - Endpoint(s)
- Or, there's HID
  - Decisions, decisions…



USB made simple

- Real USB devices are usually HID
  - Don’t need an OS driver
- If not, then they usually have
  - 1 configuration, with
    - 1 interface, with
      - 1 endpoint
      - Sometimes 2 (Biopac MP35)
- Sometimes they’re a fake serial port

PyUSB

- Python wrapper for 3 USB libraries:
  openUSB, libUSB 0.x, libUSB 1.0
  – Autodetects which is installed
  – I use libUSB 1.0 for best windows compatibility
- Procedure:
  1. Find device
  2. Set interface
  3. Read & write to your heart’s content
  4. Close (if you don’t want python to do it)

USB missile launcher example

```
import usb.core, usb.util

usb_device = usb.core.find(idVendor = 0x1941, idProduct = 0x8021)

if not usb_device:
   raise usb.core.USBError('USB missile not detected')

usb_device.set_configuration()

status = usb_device.read(0x81, 8)
```

Types of transfers

- Bulk / Interrupt
  - The usual type for bulk data
- Isochronous
  - For things that must be on time (won’t discuss)
- Control
  - For control messages, config stuff
  - Just a bulk transfer to endpoint 0x0, with some extra data fields
  - For HID devices, this is how you write to them!

PyUSB commands

- FILL IN LATER

Reverse-Engineering USB

- Some companies don’t really want you to fully enjoy your hardware
  - Windows-only?
  - Crappy drivers?
  - Too bad!

- That’s OK, we’ll make our own in python
  - But how to reverse the traffic?
- First, we must sniff

Sniff USB

- Old & krunky
- But it outputs a text log file
- Python scripts to post-process
  - Eliminating useless cruft
  - Translating hex codes to opcodes
  - Scraping hex blocks into binary files for replay attacks or hex-editing
- After processing, output corresponds to pyUSB function calls!

Sniffing demo

Specific Examples

- Biopac MP35 was tough
  - Two separate drivers required
    - Stage 1: Cypress EZ-USB chip with soft firmware
      - Sent with control transfers
    - re-enumerates as new device!
    - Stage 2: TI DSP chip with soft firmware
      - Firmware sent to endpoint 1
      - Actual operation done through endpoint 2
      - Approximately 60 different commands, many modes
  - Lots of custom python code for that one

Specific Examples

- Dream Cheeky USB missile launcher
  - The code’s already online, but it made good practice
  - HID class device
    - Control motors with control transfers
    - Read limit switch status with bulk read
    - From zero to rough driver in about 30 minutes

The recap

- Python makes it fast and easy to do serious hardware control for serial & usb devices
- Sniffing & reverse-engineering USB isn't very hard
- Did we say python is cool?

- The scripts we use for USB sniffing & log cleanup are going to be online at: http://www.cvorg.ece.udel.edu/