Reverse Engineering USB Devices

Drew Fisher

December 28, 2011

whoami

Drew Fisher (zarvox)
I maintain libfreenect, a set of reverse-engineered Kinect drivers.
http://github.com/OpenKinect/libfreenect

What we'll cover

Introduction

Motivation

USB Primer

Protocol Reverse Engineering

Vision for future

Questions

Motivation: cool new devices!

- There are USB devices out there that do (really!) neat things
- The more unique the device, the less likely that the vendor supports it with a non-Windows driver

Motivation: a compatible driver

- We want to speak the same protocol. This protocol is built atop USB.

Motivation: a compatible driver

- We want to speak the same protocol. This protocol is built atop USB.
- We need to understand the device's state transitions.

Motivation: a compatible driver

- We want to speak the same protocol. This protocol is built atop USB.
- We need to understand the device's state transitions.
- We need to understand the device's data.

Motivation: a compatible driver

- We want to speak the same protocol. This protocol is built atop USB.
- We need to understand the device's state transitions.
- We need to understand the device's data.
- So let's watch the messages that go by, and figure out which ones are which.

USB: just the basics

- Distinction between Host and Device
- All communications are started by the host
- Devices have multiple endpoints which are in effect, separate data queues

USB Primer - USB endpoint/transfer types

Four types:

- Control
- Interrupt
- Isochronous
- Bulk

USB Primer - Control Transfers

- Host starts a request, specifies request number and direction
- Either host or device transfers data
- Device or host acknowledges transfer if successful
- Every USB Device supports control transfers on endpoint 0

USB Primer - Interrupt Transfers

- Guaranteed bounds on latency
- Attempts retransmission next epoch on error
- Useful to notify host of device state change
- Example: used for Human Interface Device reports (mice, keyboards)

USB Primer - Isochronous Transfers

- Guaranteed polling rate and bandwidth
- No retransmission
- Useful for avoiding jitter - dropped packets are okay, as long as stream is realtime
- Example: used for USB Video Class video stream

USB Primer - Bulk Transfers

- Large bursty data
- CRC provides error detection
- Retransmission provides reliable delivery
- Example: USB Mass storage (disks, flash drives)

Putting it together

- Under normal operation, the host's driver tracks the device's state.
- So all information pertaining to state transitions are encoded in the transfers.
- State changes require reliable delivery.
- Streaming realtime data (like audio) does not.

So now what?

Assumption: we are working with devices that already have working drivers.
The usual workflow:

1. Obtain USB traces of normal operation
2. Stare at them until they make sense
3. Write driver

Step 1: get data

Hardware loggers:

- TotalPhase Beagle 480
- OpenVizsla – `http://openvizsla.org/`

Software loggers:

- BusDog – Windows USB filter driver `http://code.google.com/p/busdog/`
- `/dev/usbmon`

Step 2: understand data

- Download/extract TotalPhase Data Center for your platform: `http://www.totalphase.com/products/data_center/`
- Get USB trace from someone who bought a Beagle 480: `git clone git://github.com/adafruit/Kinect.git`
- Open `Kinect/USBlogs/enuminit.tdc` with Data Center
- Start reading ;)

Pattern matching

Problems developers face
Protocol versioning
Packet framentation and reassembly

Latency measurement

Pattern matching

Problems developers face	**Solution**
Protocol versioning	Magic bytes
Packet framentation and reassembly	Length/size bytes
	Sequence numbers
Latency measurement	Timestamps

Structure

Bootloader command:

```
uint32_t magic;
uint32_t tag;
uint32_t bytes;
uint32_t cmd;
uint32_t address;
uint32_t unknown;
```

Structure

Audio in transfer:

```
uint32_t magic; // 0x80000080
uint16_t channel; // Values between 0x1 and 0xa
indicate audio channel
uint16_t len; // packet length
uint16_t window; // timestamp
uint16_t unknown; // ???
int32_t samples[]; // Size depends on len
```

Step 3: write driver

libusb is pretty cool and makes prototyping easy (compared to prototyping kernel drivers).
`http://www.libusb.org/wiki/libusb-1.0`

Live demo!

What should RE tools do?

- Help human notice patterns, especially common ones
- Help human test hypotheses against larger dataset
- Help humans work together

Questions!

http://openkinect.org/