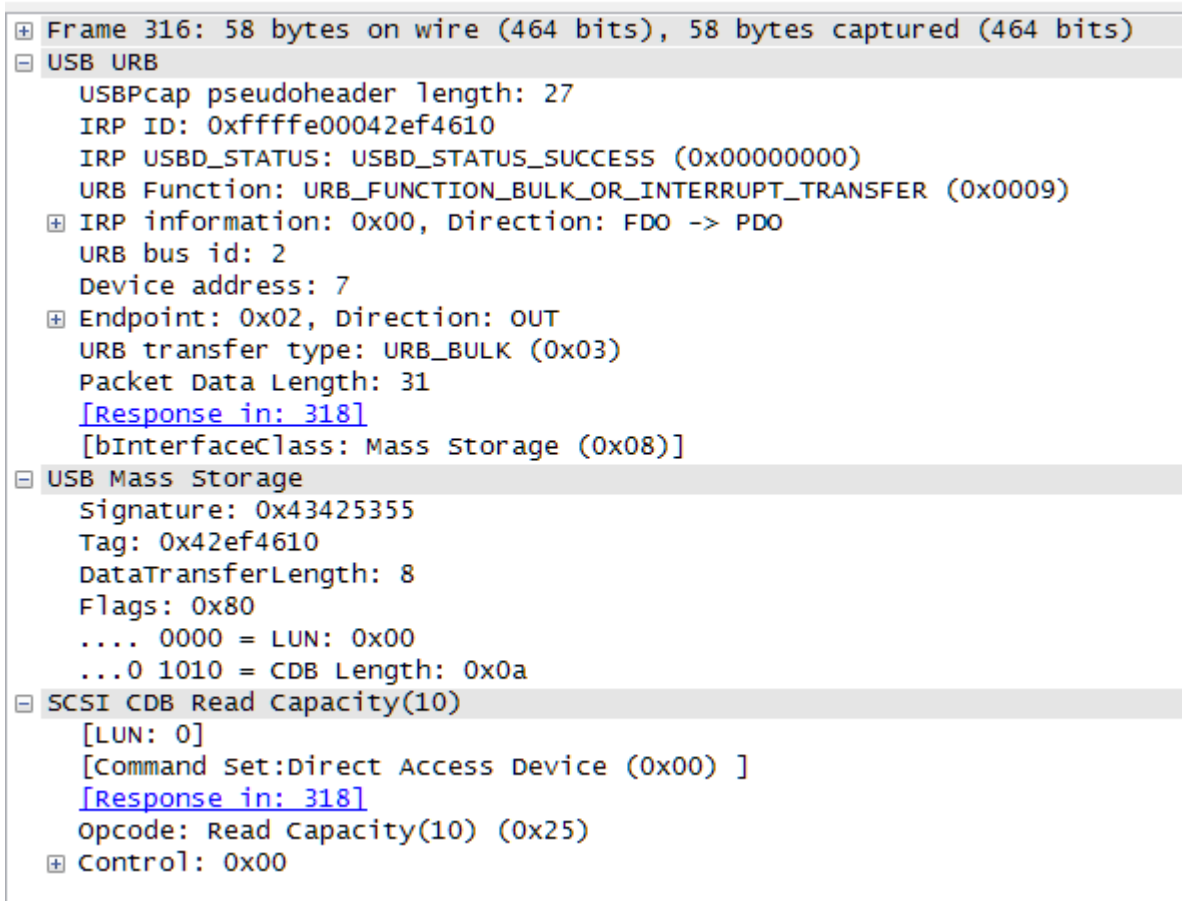
USB Basic Overview

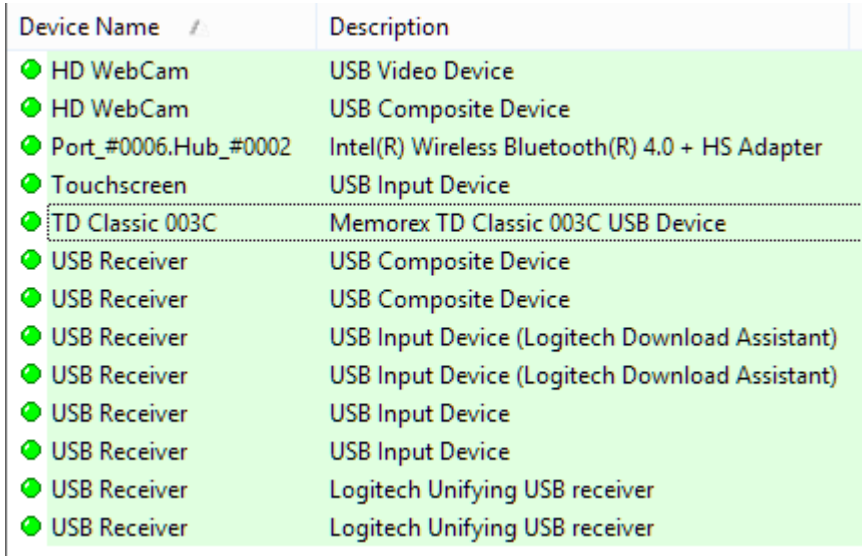
- Communications protocol built up by layers of packetized data
  - Very similar to the OSI model as covered in class
- Enumeration process connects a USB device to the PC
  - Device identifies itself and its capabilities, PC loads drivers
- Different USB devices are defined by different USB Classes
  - Mass Storage, HID, Audio, Video, Communications, Printers…
- A controller *must* exist on the device to handle USB protocol

Typical USB Packet in Wireshark



Slight Complications

- USB Hubs
  - Invisible from the host's perspective
- Internal hubs
  - Webcams, Bluetooth radios
- Composite Devices
  - Webcams with microphones



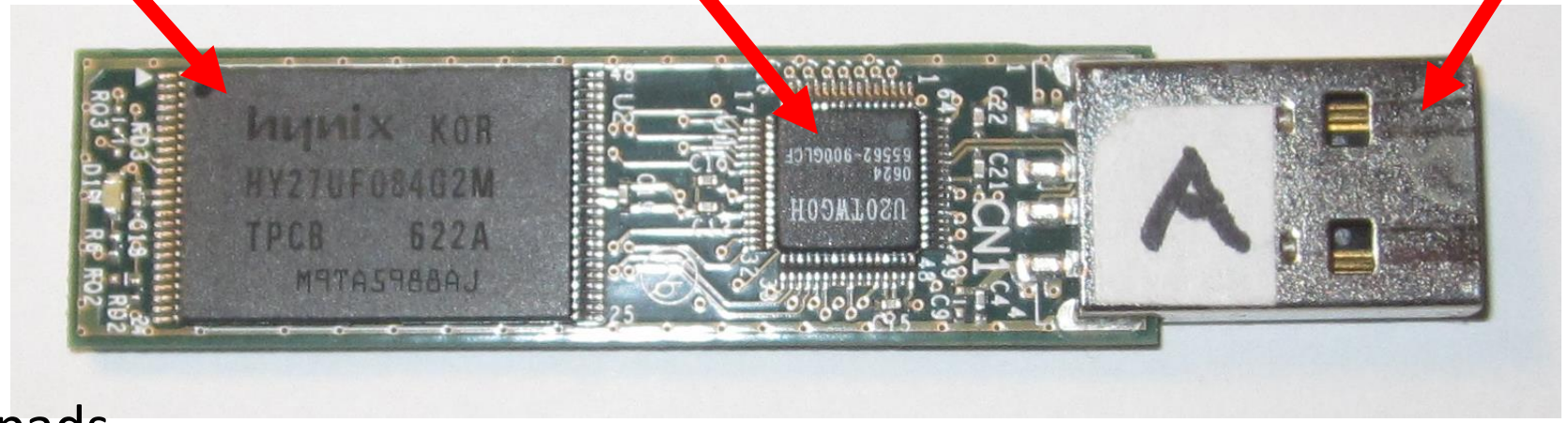
Device Name	Description
HD WebCam	USB Video Device
HD WebCam	USB Composite Device
Port_#0006.Hub_#0002	Intel(R) Wireless Bluetooth(R) 4.0 + HS Adapter
Touchscreen	USB Input Device
TD Classic 003C	Memorex TD Classic 003C USB Device
USB Receiver	USB Composite Device
USB Receiver	USB Composite Device
USB Receiver	USB Input Device (Logitech Download Assistant)
USB Receiver	USB Input Device (Logitech Download Assistant)
USB Receiver	USB Input Device
USB Receiver	USB Input Device
USB Receiver	Logitech Unifying USB receiver
USB Receiver	Logitech Unifying USB receiver

Anatomy of a USB flash drive

NAND Flash

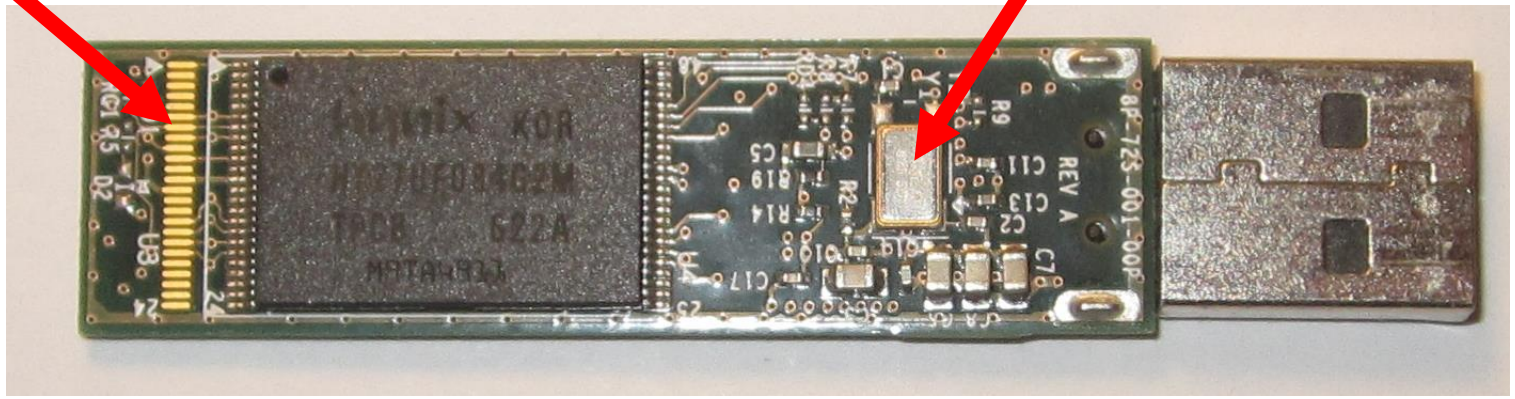
USB Controller

USB Port



Extra landing pads

USB Clock




A
B
C
D
E
F
G
H
J
Memorex
Toshiba
BU Today
Patriot
Centon
???
Silicon Power
PNY
Patriot
Sandisk
HP
8GB
Lexar
Verbatim
PNY
Sandisk
Kingston
DT101 G2
8GB

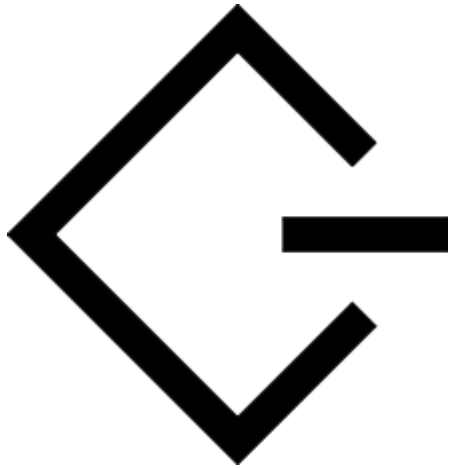
Summary of observations

- Chip markings are meaningless, most designers use ASICs
- Design of these drives is to optimize cost as much as possible
- Controller IP cores are provided by 4 main vendors: Phison, SMI, Alcor, and USBest
- Drive manufacturers have no affinity to a single controller/flash (Even within the same models)
- SanDisk is the least susceptible to reverse engineering.

Most common configuration

bDeviceClass	bDeviceSubClass	bInterfaceProtocol	SCSI Device Type
Mass Storage	SCSI Transparent Command Set	Bulk Only Transport	Direct Access Drive

- This information defines the set of documents needed to understand a USB packet
- USB Mass Storage uses SCSI
- Protocol is basically the following
  - SCSI op > Bulk Data (IN/OUT) > SCSI Status
- Vendors tend to use vendor reserved opcodes in SCSI to configure devices



Dangers of configurability

- Generic controllers are inherently reprogrammable because NAND is awful
- Lots of slack space in memory that “can’t be sold”
- What happens when this programmability is exploited? BadUSB
  - Custom firmware can be written
  - Can do anything a USB device is capable of
  - OS has an inherent trust in the USB controller doing what it’s designed to do
  - USB controller can “lie” to the OS about everything

How feasible is this?

- Vendors rely on security through obscurity
- Leaked tools are available and USB traffic can be sniffed
- Vendor tools tend to use custom SCSI commands to reconfigure devices
- Data is transferred in raw unencrypted form
- Data is not signed and is blindly accepted by the controller
- Limited by capabilities of 8051 controller and RAM.

Project Implementation

- Reverse engineered an SMI SM3257-ENAA controller found in BU Today flash drive and PNY Apache drive
- Program capable of reading writing the controller's identification (CID) section of the firmware by replaying SMI's vendor commands
- Can spoof VID, PID, Serial Number, Vendor Name, Manufacturer Name, and Product Name
- Given more time/resources this could be developed into a more potent weapon