OPENING BLACK BOX SYSTEMS WITH GREATFET+FD

TROOPERS18
KATE TEMKIN & DOMINIC SPILL
WHO WE ARE

Kate Temkin (@ktemkin):
• slayer of Tegras, destroyer of worlds
• glitch witch & tool-builder
• educational (reverse) engineer

Dominic Spill (@dominicgs):
• cannot stop being extraordinary, on penalty of deportation
• shark whisperer & demo dancer
MANY THANKS TO

• Travis Goodspeed (@travisgoodspeed)
• Sergey Bratus (@sergeybratus)
• Michael Ossmann (@michaelossmann)

PEOPLE WHO GIVE US MONEY

• Great Scott Gadgets (@gsglabs)
Why target USB?

USB is everywhere.
WHY USB?

The capability to monitor, MITM, & emulate USB devices enables:

- **Understanding the behaviors of** USB and driver stacks
- **Building tools** that work with existing hardware / software
- Building implants and tools for **playing NSA**.
- One **to get a foot in the door** for understanding black box systems.
WHY PROXY?

All too often-- as with black box systems-- we don’t control the host software stack:

- Game consoles [e.g. the Nintendo Switch]
- In car entertainment [e.g. Tesla consoles]
- Point of sale
- Televisions

- ... pretty much any embedded device that can act as a USB host!
USBProxy is a tool that allows us to **proxy the connection** between a USB host and device. While proxying a connection we can:

- Log USB packets (**cheap protocol analysis**)
- **Modify data** being sent to or received from a device
- **Inject new packets** into the connection, or **absorb packets**
- **Capture side-channel information and precisely time glitching attacks**

Original version was based on a BeagleBone Black in C++. We’ve rewritten it to take advantage of FaceDancer’s more granular control.
[let’s monitor some USB]
USB CLASSES

In addition to specifying the standard protocol used for enumeration/configuration, the specs also specify protocols for **standard device classes**, allowing e.g. operating systems to provide **standardized drivers**.

- Human Interface Device (keyboards, mice, **datagloves**; the usual)
- Serial (e.g. CDC-ACM)
- Mass storage (UMS bulk only / UAS)
- Audio / Video
- Midi
- Scanners
- Networking
- etc.
[let’s slack off]

https://github.com/ktemkin/Facedancer/blob/master/usbproxy-switch-invertx.py
EXPLORATORY RE

There are many USB hosts and devices for which firmware isn’t easily available—but we don’t always need firmware to do interesting things to a system.

- Can we discover behaviour?
- Find firmware functions?
- What about identifying hosts?
EXPLORING FUNCTIONALITY

By monitoring and modifying USB packets we can discover functionality of a host system

- Does it take firmware updates via USB?
  - What filename is it looking for?
  - Does it read that file multiple times?
- How does the host enumerate the device?
  - Order and length of requests
  - Timing
  - Windows Compatibility ID
  - umap2 already does this, let’s port it to new FaceDancer
EXPLORING FUNCTIONALITY

By monitoring and modifying USB packets we can discover functionality of a host system

- Does it take firmware updates via USB?
  - What filename is it looking for?
  - Does it read that file multiple times?
- How does the host enumerate the device?
  - Order and length of requests
  - Timing
  - Windows Compatibility ID
  - umap2 already does this; let’s port it to new FaceDancer
- What are the device’s access patterns?
[let’s run a simulated firmware update]
Of course, nothing says our emulated devices have to behave nicely.

**Example:** most systems assume that disk contents *don’t change on their own*

**Reality:** in practice, *they totally can*

**Example firmware update sequence:**

- USB host reads firmware off flash drive, computing a checksum as it does
- USB host verifies the checksum, which passes
- USB host *rereads the firmware and flashes it to ROM*
[let’s fetch... *twice*]
EXPLORING FUNCTIONALITY

By monitoring and modifying USB packets we can discover functionality of a host system

- Does it take firmware updates via USB?
  - What filename is it looking for?
  - Does it read that file multiple times?
- How does the host enumerate the device?
  - Order and length of requests
  - Timing
  - Windows Compatibility ID
  - umap2 already does this, let’s port it to new FaceDancer
let’s talk about firmware filenames
GlitchKit

- Synchronization Features
  - Event Routing
  - Clock Management
- Stimulus Generation
  - USB Host
  - USB Device
  - eMMC Device (not yet complete)
- Triggering Features
  - Simple Event Triggers
  - UART Triggers
  - Trigger Output
gf = GreatFET()
gf.switch_to_external_clock()
gf.glitchkit.provide_target_clock(VBUS_ENABLED);

gf.glitchkit.simple.watch_for_event(1, [('EDGE_RISING', 'J1_P7')])
gf.glitchkit.use_events_for_synchronization(COUNT_REACHED)

gf.glitchkit.trigger_on_events(HOST_SETUP_TRANSFER_QUEUEUED)
gf.glitchkit.usb.capture_control_in(request=GET_DESCRIPTOR,
    value=GET_DEVICE_DESCRIPTOR_DESCRIPTOR, length=18)
QUESTIONS?
THANKS FOR LISTENING!

JOIN US:
https://github.com/greatscottgadgets/greatfet
https://github.com/ktemkin/Facedancer
https://github.com/glitchkit