SHARKFEST '13

Wireshark Developer and User Conference

Expanding Wireshark Beyond Network Interfaces

Mike Kershaw, Kismet Wireless

Mike Ryan, iSEC Partners

About Us

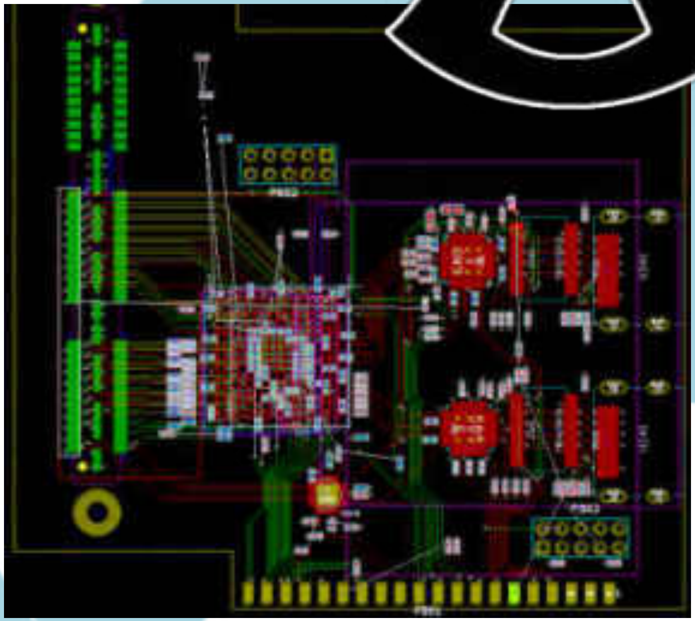
Mike Kershaw

Kismet wireless sniffer

Various open-source hardware for sniffing

Kisbee Zigbee sniffer

Daisho wired protocol sniffer



About Us

Mike Ryan

Infosec Consultant @ iSEC Partners

Bluetooth LE Researcher

2 cool 4 skool

MEGA DISCLAIMER: I speak only for myself and not my employer. I'm lucky they let me take work off today.

Motivations

- Wireshark is an amazing tool with decoders for a lot of protocols
- Open Source Hardware has seen a great boom recently
- All sorts of interesting things out there which capture packets, but which are not network interfaces
- How do we bring these into the fold easily?

Requirements

- Developer simplicity - If it's a huge pain to add Wireshark support to 3rd-party projects, it just won't happen
- Multi-platform support - We don't want to reduce Wireshark's cross-platform functionality
- Ease of use - It needs to make sense to end users!
- Security - Don't compromise privsep

Wireshark Today

- Captures from network devices
- Loads from pcap files
- Network-centric (obviously)
- Able to handle non-Ethernet traffic already (Wi-Fi, TokenRing, USB, other esoterica)
- Still needs to be a network interface or a file

Non-Network Options Today

- Log to a file, open in Wireshark
  - Not real-time, kind of annoying
- Play games with tun/tap network devices and clone packets into a virtual netdev
  - Requires root to manipulate interfaces, somewhat complex, not cross platform at all
- Write to a pipe
  - Best option so far, annoying for end users

Where we need to get to

- Don't break capturing from network devices
- Don't force compiling plugins directly into Wireshark
- MAKE IT EASY. People doing random custom projects won't spend a lot of time
- Present a standard Wireshark UI - if it's unusable or opaque it's worthless

Hurdles to External Capture

- Wireshark & Pcap like network interfaces
- All network interfaces are configured the same way (more or less)
- Running arbitrary binaries is *really scary* from a security standpoint
- Things that don't act like network devices need weird configs

Solutions!

- Wireshark (and dumpcap) can read from pipes!
- Pipes are multiplatform!
- Making a simple configuration grammar lets us define custom UI elements
- Placing responsibility for privilege escalation with the capture binary solves security issues
- Minimal changes to Wireshark internal code

Basic Extcap Architecture

- Each external capture 'plugin' is an executable provided by capture tool developers
- Don't care what language it's in
- Responds to a set of basic arguments to list interfaces, config options, and initiate capture
- Writes to a named pipe fed to dumpcap
- Basic config grammar describes UI

Extcap security

- Extcaps are launched by Wireshark - no more initial privs than the starting user
- Extcap privs are controlled by whatever provided the extcap - if it needs suidroot, they can grant that. We can't know if they do, and don't grant it
- Config grammar is non-turing, just markup

Extcap Grammar

- [type] {[attribute]=[value]}*
- Each type is a sentence
- Extremely simple to generate - designed to be easy to add to tools, generate from printf
- Simple to parse - non-evaluated, non-escaped, non-turing

Interface sentences

- Interface sentences list known interfaces for each extcap, and a user-displayable interface name as well as the calling value passed back to extcap
- Interfaces make up the list of supported interfaces in Wireshark

```
interface {display=Interface One}{value=int1}
interface {display=Interface Two}{value=int2}
```

Multiple Interfaces

- Multiple interfaces can be supported by a single extcap plugin (same as multiple Ethernet devices)
- Each interface can have independent configs and will spawn an independent extcap capture
- Extcap plugin provide a list of interfaces, allowing for searching USB, remote network, etc

DLT sentences

- Extcap tools need to tell Wireshark what DLTs are supported on a capture
- Provides DLT#, name, and displayable field

```
dlt {number=147}{name=USER0}{display=Bluetooth Low Energy}
```

DLT = Data Link Type
Specifies Link Layer

Arguments

- Most complex function to handle
- Can be presented to the user as several types; int, double, etc text fields, boolean checkboxes, checkbox lists
- Can also be populated GTK types like selector or radio buttons
- Allows for tooltips for explanation

Arguments

- Each argument has a 'call=' argument, which is the literal call made to the extcap binary
- Can be 'call=--longarg' or 'call=-a'
- 'type=' determines how it is presented in Wireshark
- Selector/Radio/Check selectors are populated with additional 'value' sentences

Arguments (examples)

```
arg {number=0}{call=frequency}{display=Frequency}
    {type=integer}{range=2400,2480}{default=2437}
    {tooltip=Frequency in MHz, 2400-2480}

arg {number=1}{call=hop}{display=Boolean}
    {type=boolean}{default=true}
    {tooltip=Dynamically hop channels}
```

Values

- Multiple *value* sentences can be associated with an argument
- Pre-fills selectables or radio button groups
- Whatever the user selects will be passed to the argument's call

```
arg {argnum=0}{value=12345}{display=First}
```

Calling

- Take each 'arg' sentence
- Build an argument list of the arg calls
- Run extcap binary pointing to the FIFO

```
some_extcap --call1=foo --call2=bar --call3=1000000 --
fifo=/tmp/excap12234324
```

Error checking

- We want to do as much as possible to make it hard for the user to screw up
- Since we're targeting esoteric hardware we want to handle esoteric arguments
- Transparently encode scientific notations (frequency of 100e6)
- Range checking can happen in the UI

External capture tools: Requirements

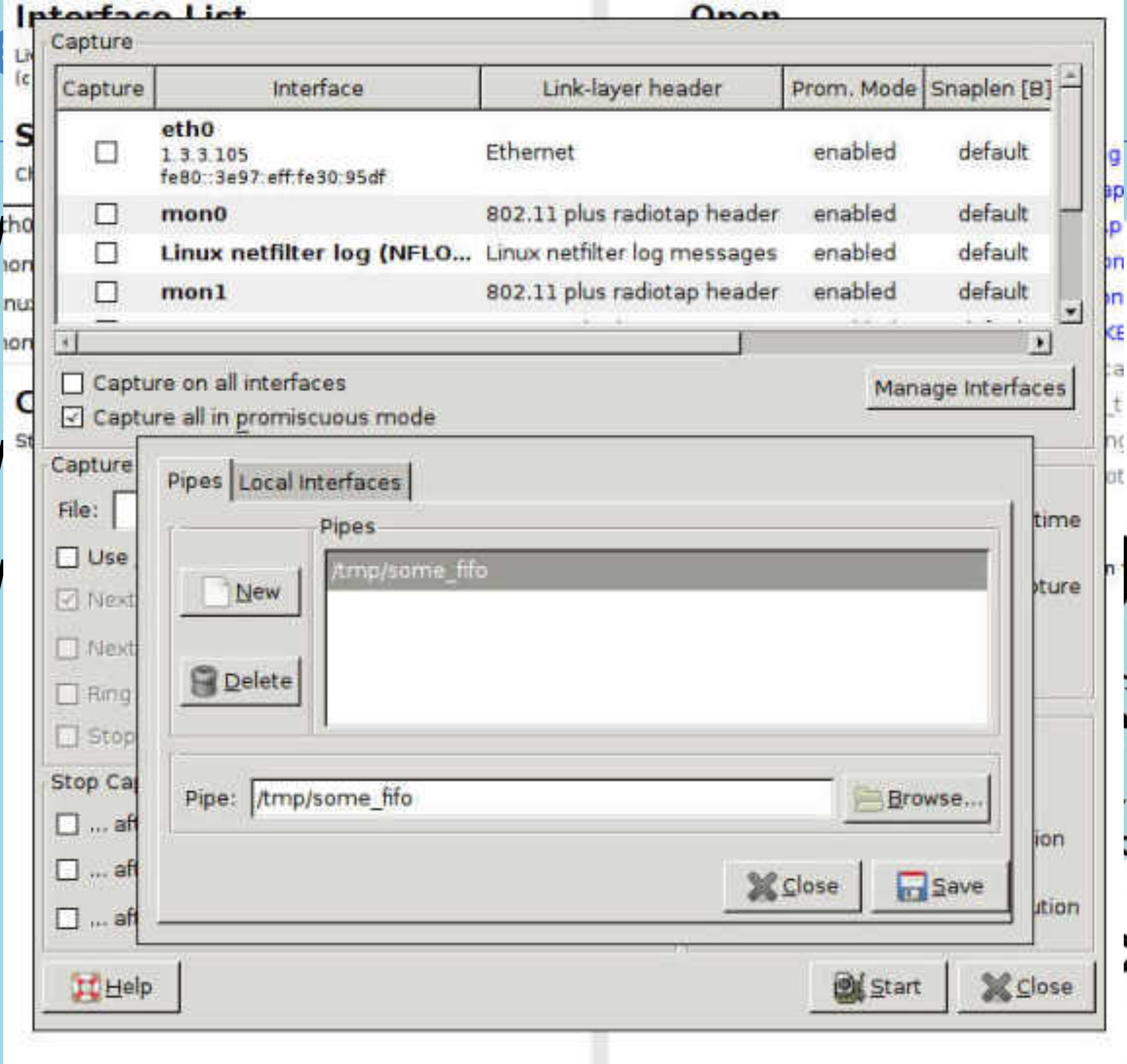
- Must respond to a handful of arguments
- Must be able to write a pcap stream to a named pipe
- Must flush pipe after each packet
- ...
- That's about it!

Wireshark Pipes

- What did we change?
  - Not much!
- Wireshark has had pipes since like forever
- We just slap a nice[r] GUI on it
  - mumble mumble DLTs and exec'ing extcaps

Wire

- W
- W
- W



Interface List

Open

Capture

Capture	Interface	Link-layer header	Prom. Mode	Snaplen [B]
□	**eth0** 1.3.3.105 fe80::3e97:eff:fe30:95df	Ethernet	enabled	default
□	**mon0**	802.11 plus radiotap header	enabled	default
□	**Linux netfilter log (NFLO...**	Linux netfilter log messages	enabled	default
□	**mon1**	802.11 plus radiotap header	enabled	default

□ Capture on all interfaces

☑ Capture all in promiscuous mode

Capture

File:

□ Use

☑ Next

□ Next

□ Ring

□ Stop

Stop Ca

□ ... af

□ ... af

□ ... af

Pipes | Local Interfaces

Pipes

/tmp/some_fifo

Pipe: /tmp/some_fifo

Ye Olde Way

- Call `dumpcap -D` to get all interfaces
- Call `dumpcap -L` to get DLTs from interface
- Select options from static GUI
- Pass args into `dumpcap` for capture

Everything boils down to `pcap_` calls:
Wireshark, dumpcap, and libpcap all need to be taught new interfaces! LAME

NEW! And Improved!

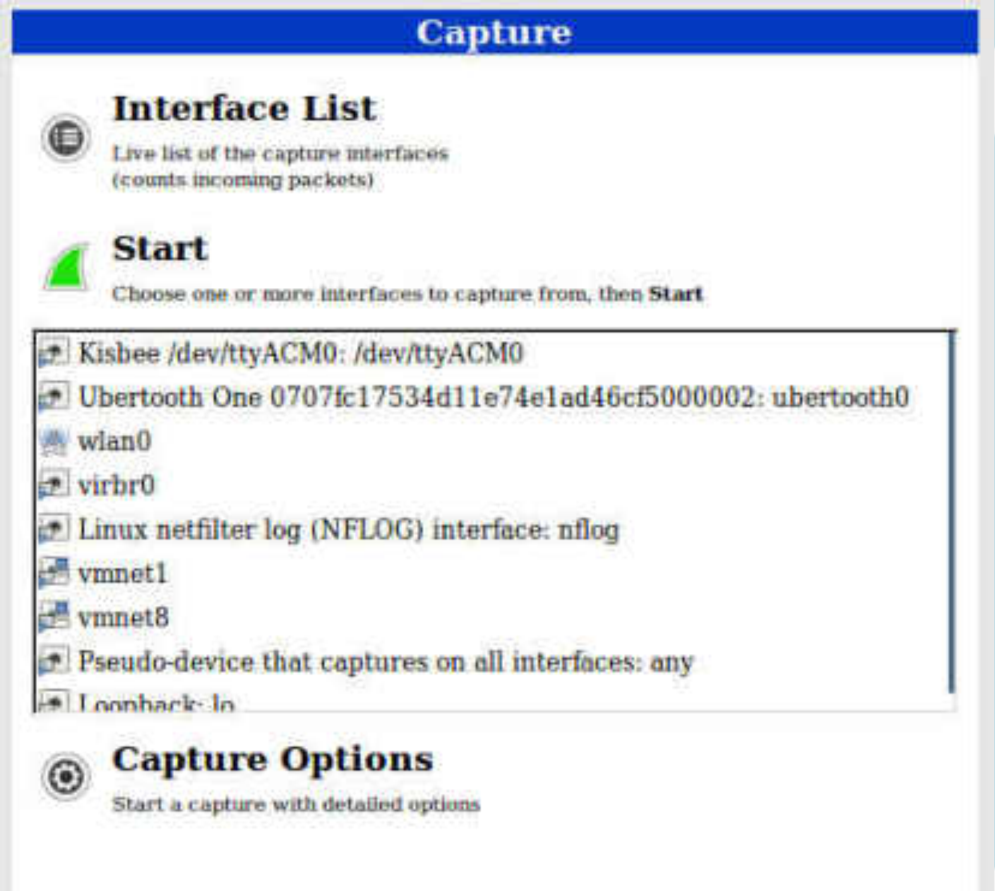
- Call `dumpcap -D` to get all PCAP interfaces
  - For each extcap: `extcap --list-interfaces`
- Call `dumpcap -L` to get DLTs from PCAP interface
  - `extcap --list-dlts --interface foo123`
- Select options from static GUI and dynamic GUI
  - `extcap --config --interface foo123`
- Pass args into `dumpcap` for capture
  - `extcap --capture --fifo /tmp/ex898 ...`
  - `dumpcap -i /tmp/ex898 <- pipe!`

Demo!

Either you just saw something awesome, or you just saw us scramble and fail!

Maybe both?

Demo!



Capture

Interface List

Live list of the capture interfaces
(counts incoming packets)

Start

Choose one or more interfaces to capture from, then **Start**

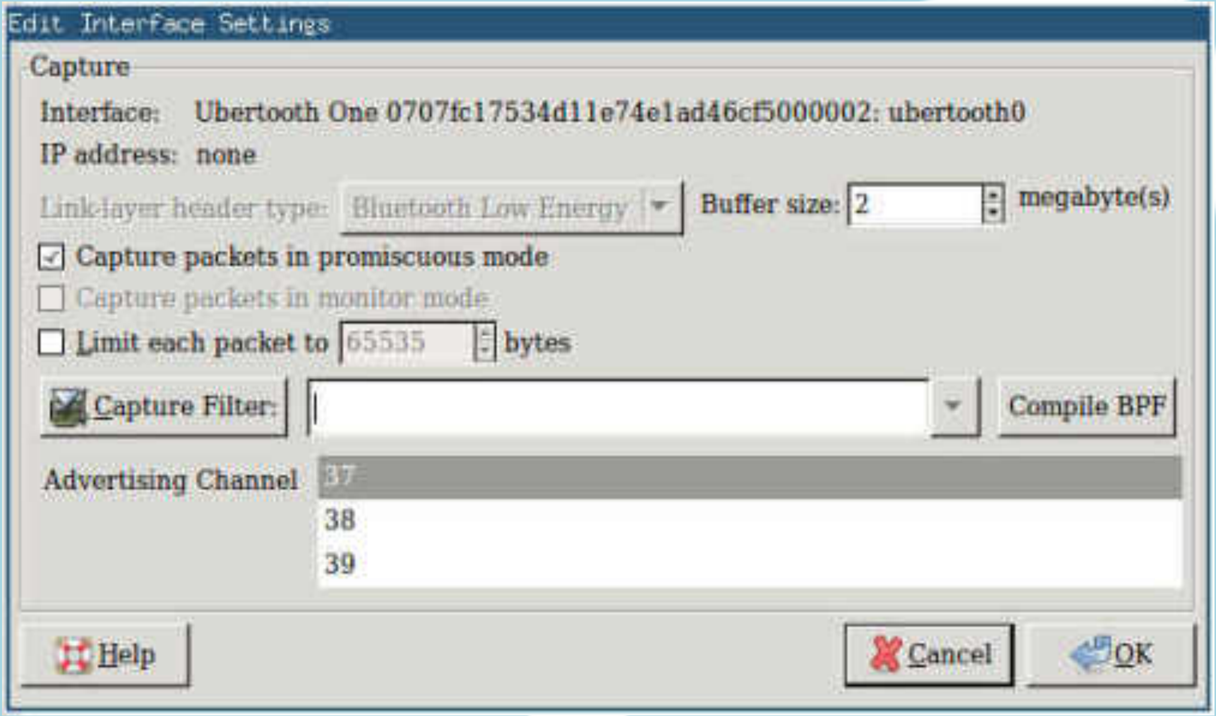
Kisbee /dev/ttyACM0: /dev/ttyACM0
Ubertooth One 0707fc17534d11e74e1ad46cf5000002: ubertooth0
wlan0
virbr0
Linux netfilter log (NFLOG) interface: nflog
vmnet1
vmnet8
Pseudo-device that captures on all interfaces: any
Loopback: lo

Capture Options

Start a capture with detailed options

Demo!

Demo!



Edit Interface Settings

Capture

Interface: Ubertooth One 0707fc17534d11e74e1ad46cf5000002: ubertooth0

IP address: none

Link-layer header type: Bluetooth Low Energy

Buffer size: 2 megabyte(s)

☑ Capture packets in promiscuous mode

□ Capture packets in monitor mode

□ Limit each packet to 65535 bytes

Capture Filter:

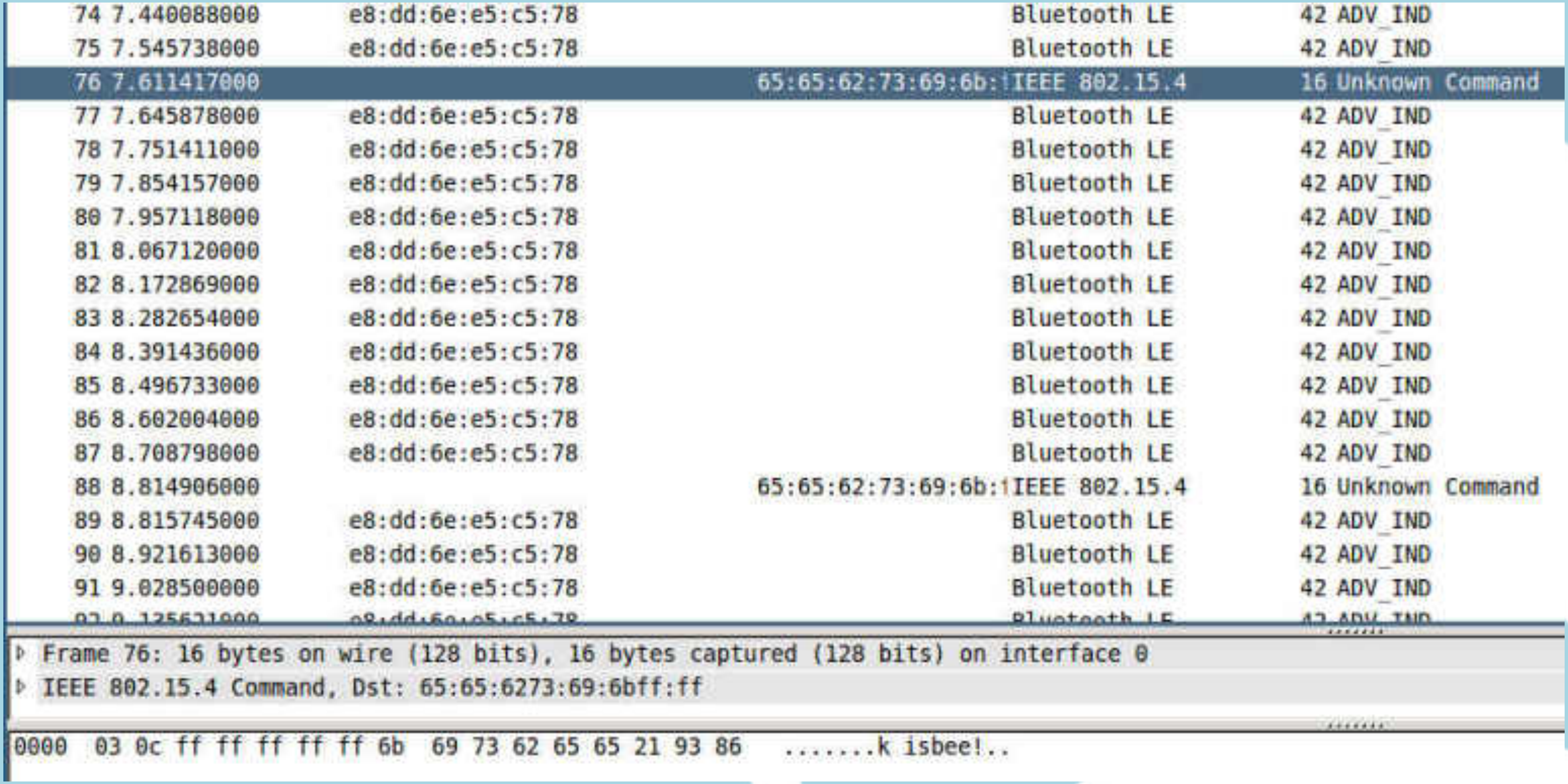
Advertising Channel

37

38

39

Demo!



```
 74 7.440088000   e8:dd:6e:e5:c5:78                      Bluetooth LE         42 ADV_IND
 75 7.545738000   e8:dd:6e:e5:c5:78                      Bluetooth LE         42 ADV_IND
 76 7.611417000                       65:65:62:73:69:6b:!IEEE 802.15.4        16 Unknown Command
 77 7.645878000   e8:dd:6e:e5:c5:78                      Bluetooth LE         42 ADV_IND
 78 7.751411000   e8:dd:6e:e5:c5:78                      Bluetooth LE         42 ADV_IND
 79 7.854157000   e8:dd:6e:e5:c5:78                      Bluetooth LE         42 ADV_IND
 80 7.957118000   e8:dd:6e:e5:c5:78                      Bluetooth LE         42 ADV_IND
 81 8.067120000   e8:dd:6e:e5:c5:78                      Bluetooth LE         42 ADV_IND
 82 8.172869000   e8:dd:6e:e5:c5:78                      Bluetooth LE         42 ADV_IND
 83 8.282654000   e8:dd:6e:e5:c5:78                      Bluetooth LE         42 ADV_IND
 84 8.391436000   e8:dd:6e:e5:c5:78                      Bluetooth LE         42 ADV_IND
 85 8.496733000   e8:dd:6e:e5:c5:78                      Bluetooth LE         42 ADV_IND
 86 8.602004000   e8:dd:6e:e5:c5:78                      Bluetooth LE         42 ADV_IND
 87 8.708798000   e8:dd:6e:e5:c5:78                      Bluetooth LE         42 ADV_IND
 88 8.814906000                       65:65:62:73:69:6b:!IEEE 802.15.4        16 Unknown Command
 89 8.815745000   e8:dd:6e:e5:c5:78                      Bluetooth LE         42 ADV_IND
 90 8.921613000   e8:dd:6e:e5:c5:78                      Bluetooth LE         42 ADV_IND
 91 9.028500000   e8:dd:6e:e5:c5:78                      Bluetooth LE         42 ADV_IND
 92 9.135621000   e8:dd:6e:e5:c5:78                      Bluetooth LE         42 ADV_IND

▹ Frame 76: 16 bytes on wire (128 bits), 16 bytes captured (128 bits) on interface 0
▹ IEEE 802.15.4 Command, Dst: 65:65:6273:69:6bff:ff

0000  03 0c ff ff ff ff ff 6b  69 73 62 65 65 21 93 86   .......k isbee!..
```

What needs finishing

- Better error handling
- Killing off opened processes better
- Testing on Windows
- Enforcing range & type in UI

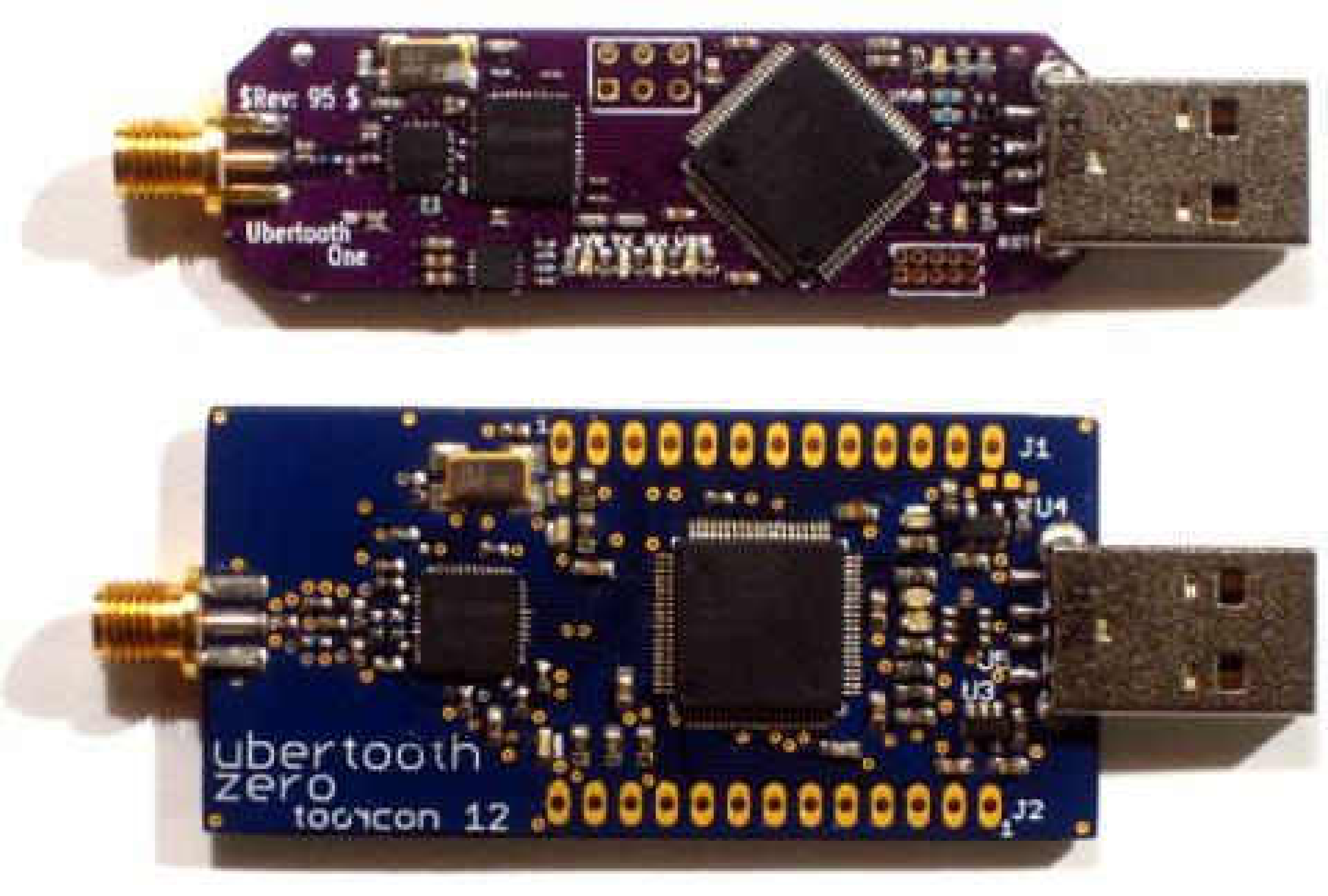
Projects!

Projects we've already started converting to extcap, or which we plan to use extcap in

Ubertooth One

- Bluetooth sniffing hardware designed by Mike Ossmann
- Bluetooth sniffing is pretty hard - you can't sniff it using commodity Bluetooth hardware
- Allows for baseband capture of Bluetooth and Bluetooth LE

Ubertooth One



Ubertooth System Interface

- Presents stream of radio data to the OS
- "Drivers" written in LibUSB, a userspace interface
- Code on OS looks for start of Bluetooth frames
- Able to generate pcaps but not emulate a device

* This is classic Bluetooth

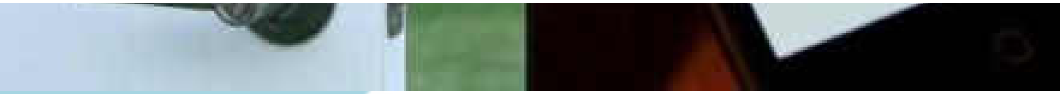
Ubertooth One Bluetooth Low Energy

- BTLE / Smart / 4.0 is way simpler than classic BT
- Which means we can actually sniff it!
- Used in some interesting places

Ubertooth One Bluetooth Low Energy

- BTLE / Smart / 4.0 is way simpler than classic BT
- Which means we can actually sniff it!
- Used in some interesting places

I'M NOT PICKING ON THESE VENDORS GOSH IT'S AN EXAMPLE



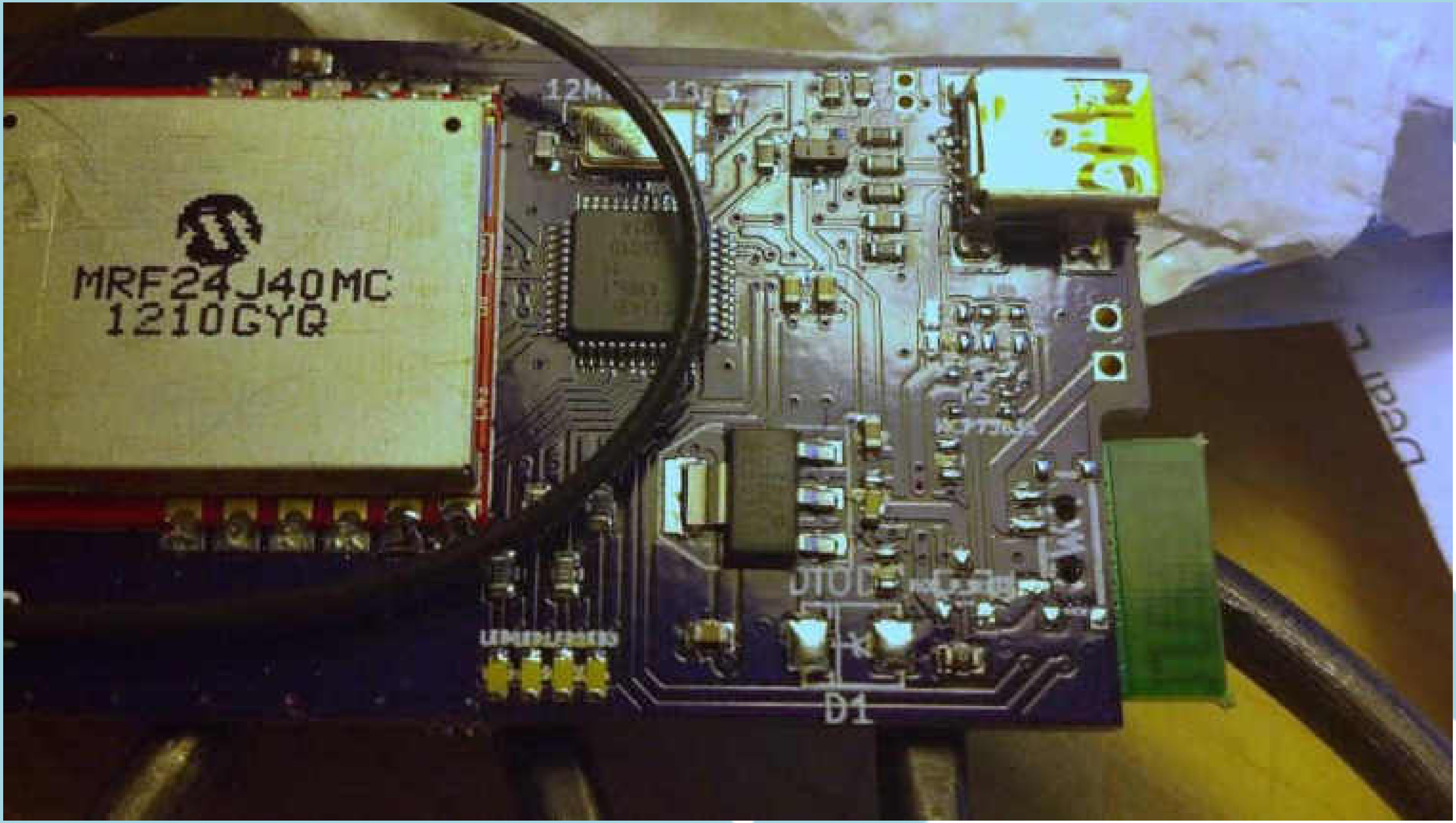
Ubertooth One BTLE: extcap

- ~100 lines of Python
- 50 of that is handling getopt(!)
- Wrapper around existing PCAP support

Kisbee

- 802.15.4 sniffer, OSHW
- Interfaces over Bluetooth SPP/RFComm or CDC-ACM serial
- Presents to OS as a USB attached serial, definitely not a network device

Kisbee



Interfacing Kisbee

- Simple (relatively) python script using PySerial talks the Kisbee protocol
- Already had support for writing to pcap files (shoehorned via Scapy)
- Protocol parser for Kisbee about ~350 lines of python

Converting Kisbee to Extcap

- Throw some ArgParser code on to handle the extcap arguments
- Do some validation of serial interfaces
- Accept --fifo instead of --file
- Add some pcap.flush() calls
- ...
- That's it! Less than 100 lines of changed code!

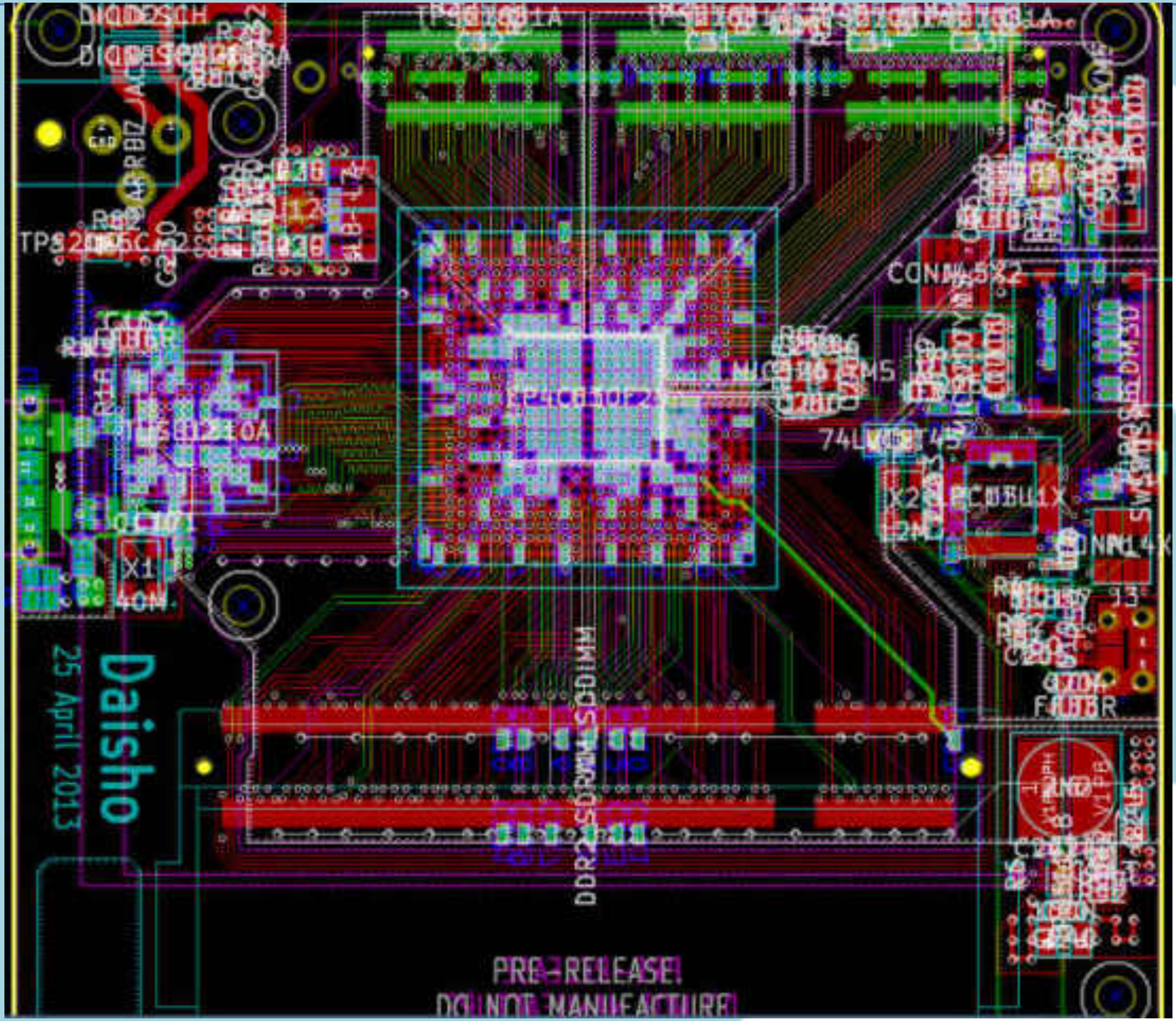
Project Daisho

- Darpa Cyber Fast Track funded, Mike Ossmann / Great Scott Gadgets principle
- Multiple wired phy-layer capture devices using a common USB3 control board
- First open-source USB3 stack (as far as we know)
- Multiple network-y devices, but not presented as network interfaces

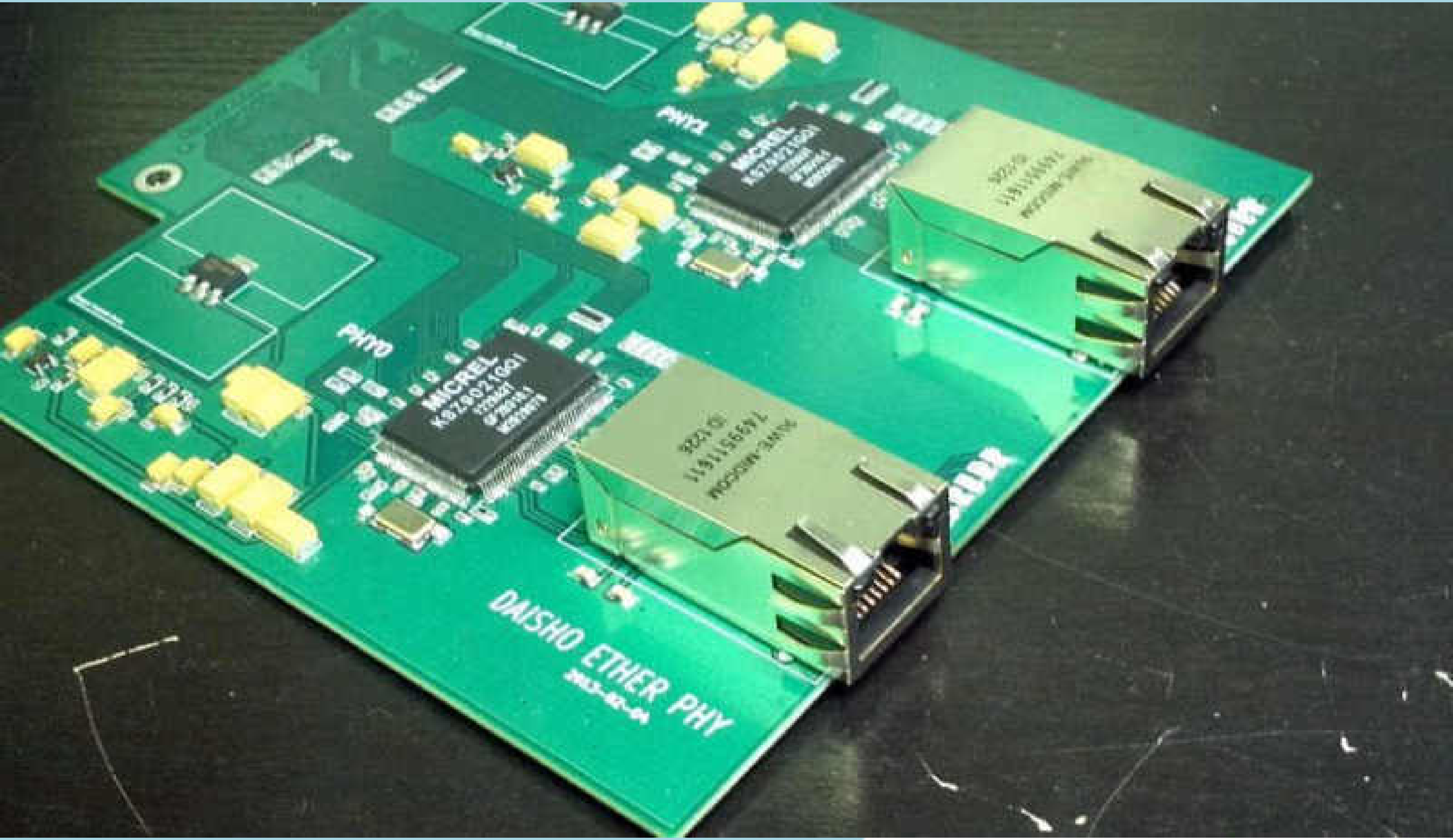
Daisho Passthrough Taps

- Gbit Ethernet
- USB3
- HDMI
- RS232
- SDR? Maybe in the future...

Daisho Mainboard



Daisho Gig-E


PHY1
MICREL KSZ9021GQI
PHY0
MICREL KSZ9021GQI
DAISHO ETHER PHY

Daisho System Interface

- Captures phy-layer data from different types of interfaces
- Wireshark already has some USB decoders, and of course Ethernet
- Lets us plug USB3 dumper code straight into Wireshark with pipes instead of huge pcap files

Software Defined Radio

- Antenna + Digitizer + Processing
- All the digital signal processing is done on the host computer, not in a dedicated IC
- Able to decode any protocol it's able to receive... in theory
- Very expensive in terms of power and compute resources, but very flexible

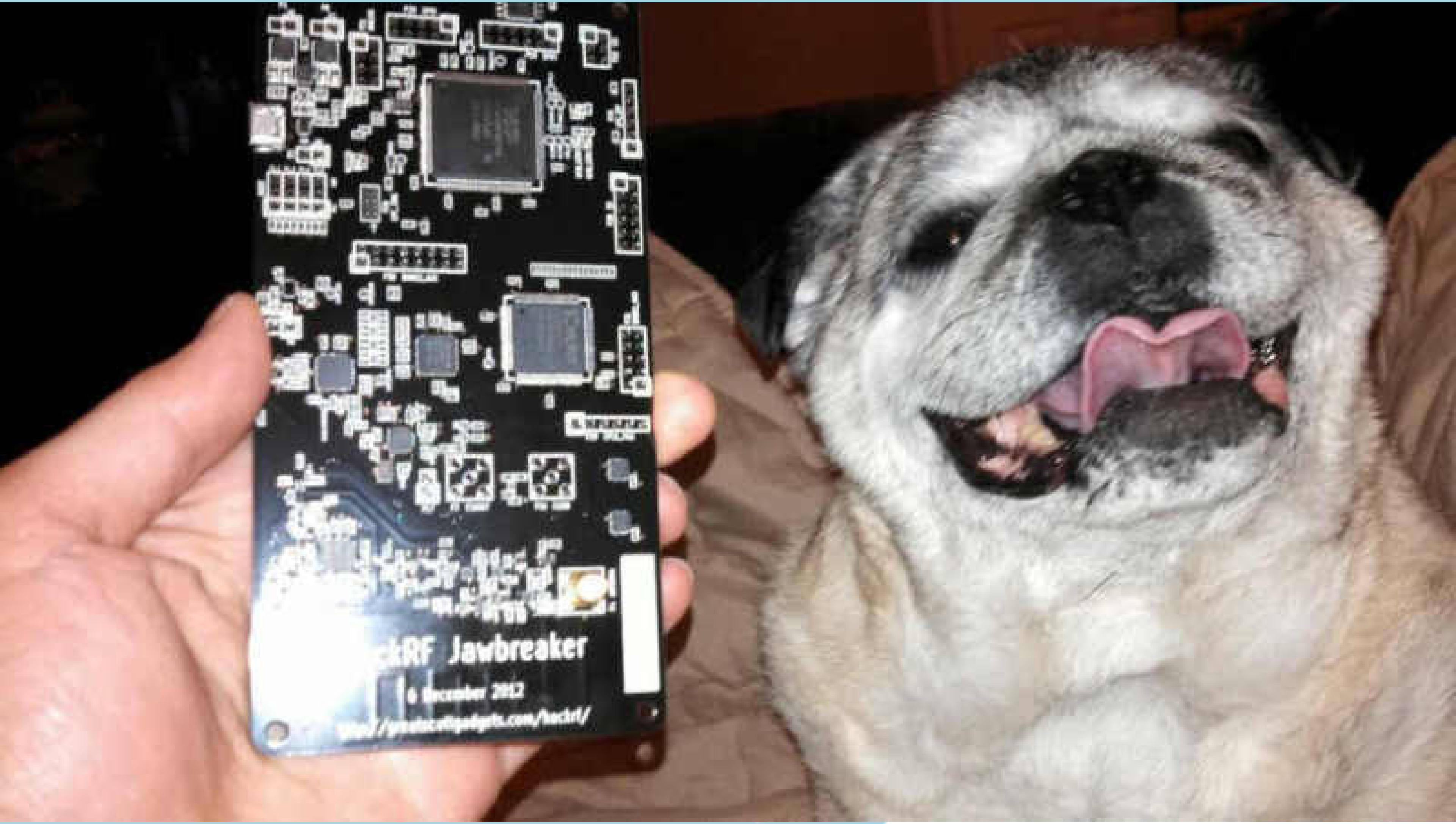
Software Defined Radio

- SDR hardware used to be extremely expensive and rare
- Recently (in the last 6 months) it's become nearly a commodity
- Software is lagging but will soon catch up now that hardware is readily available

HackRF

- Mike Ossmann / Great Scott Gadgets is making a low-cost high-flexibility SDR
- Herald of more work in SDR
- Very difficult to make a SDR work like a network interface, but now we don't have to
- 30MHz to 6GHz (!!), 20MHz samples
- In beta now, ~$400 when released

HackRF ... packets smell like bacon



RTL-SDR

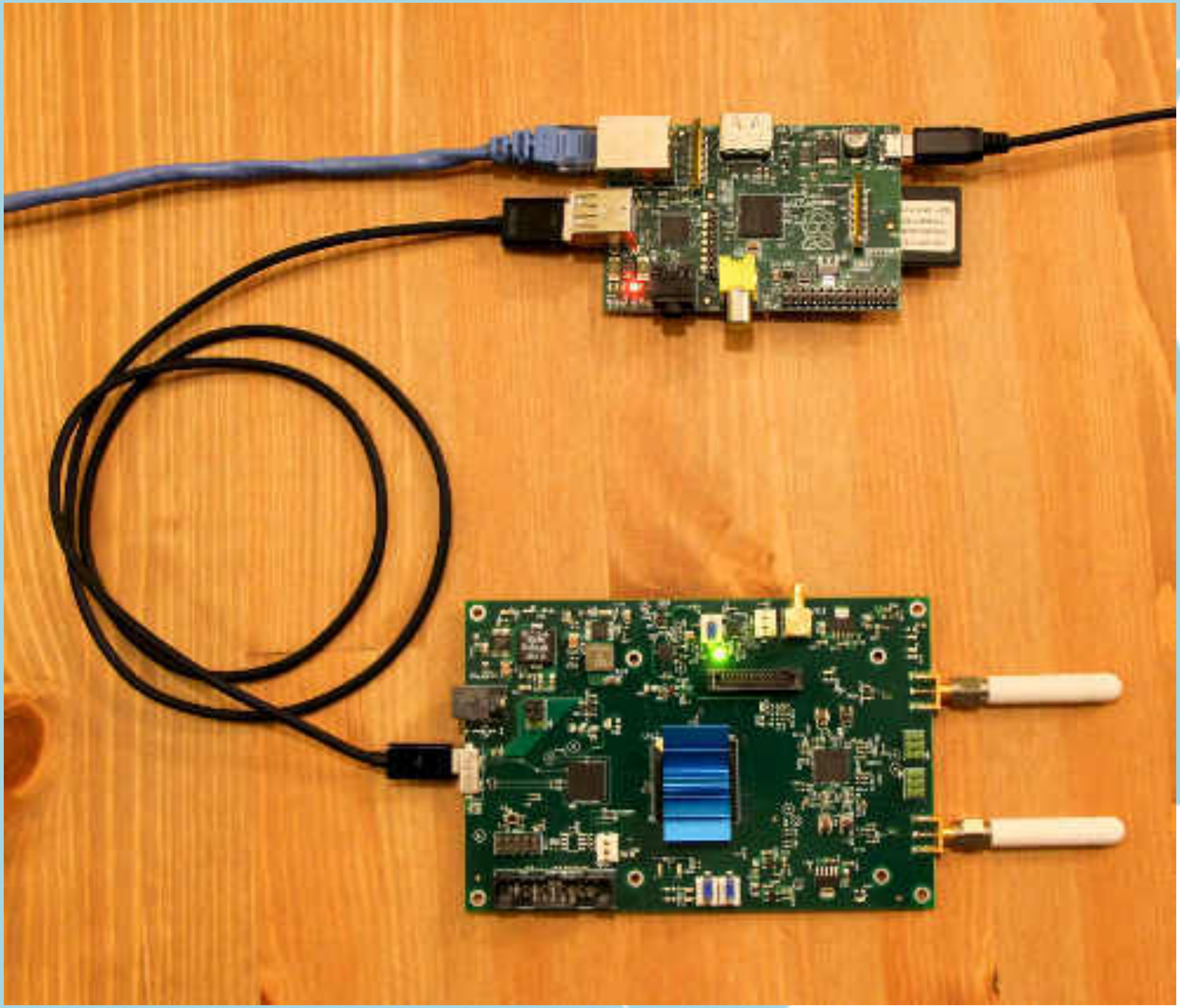
- $20 DVB tuner
- Can return proper IQ data
- 60MHz to 2.2GHz, with gaps
- Kind of crappy, but REALLY REALLY cheap
- Sufficient to capture a LOT of protocols previously not accessible with cheap hardware

RTL SDR

BladeRF

- Kickstarter, shipping w/in weeks
- 300MHz to 3.8GHz
- 40MHz capture bandwidth (!!)
- $400

BladeRF



GNU Radio

- OSS SDR radio software
- Designed as multiple pluggable blocks
- "Trivial" to chain decoder blocks and export to a pcap file
- If it's a pcap file, we can turn it into a pipe
- Student project in works to demonstrate 802.11 via GnuRadio, connected to Wireshark

SDR Decoders

- ADS-B / ACARS airplane data
- 802.11 Wi-Fi
- 802.15.4 Zigbee
- POCSAG/FLEX pager networks
- Satellite comms

- If it talks wireless in packets, it's a target

Recap

- Simple config grammar to build UIs
- Easy to write tools
- We'll be coordinating a patch to git soon after the con once we do a little cleanup
- Anything that isn't a kernel netif should work through extcap

title

- stuff