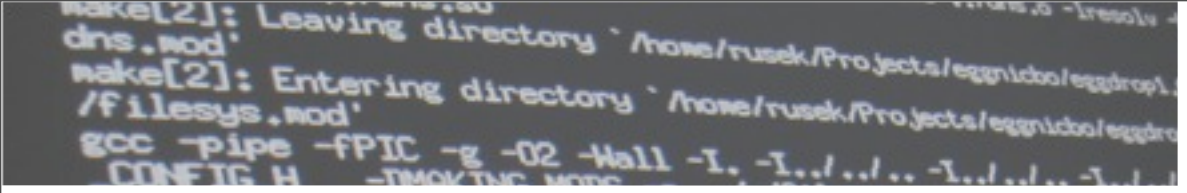


Active 802.11 fingerprinting

Sergey Bratus
Cory Cornelius, Daniel Peebles,
Axel Hansen

Dartmouth College
INSTITUTE FOR SECURITY TECHNOLOGY STUDIES
Cyber Security and Trust Research & Development
http://www.ISTS.dartmouth.edu

How it started

Toorcon 7, Johnny Cache "Virtual WLANs" talk

Different clients responded differently to changed BSSID in Authentication Resp. and Association Resp. frames

[Some clients got into endless loops]

"You are in a maze of twisty little passages implementations, all slightly different."

--TCP/IP stacks in 1990s?

The problem

Initially, an AP is just a **MAC address**
(and other easily faked info)

That's all we know.

Trust me!

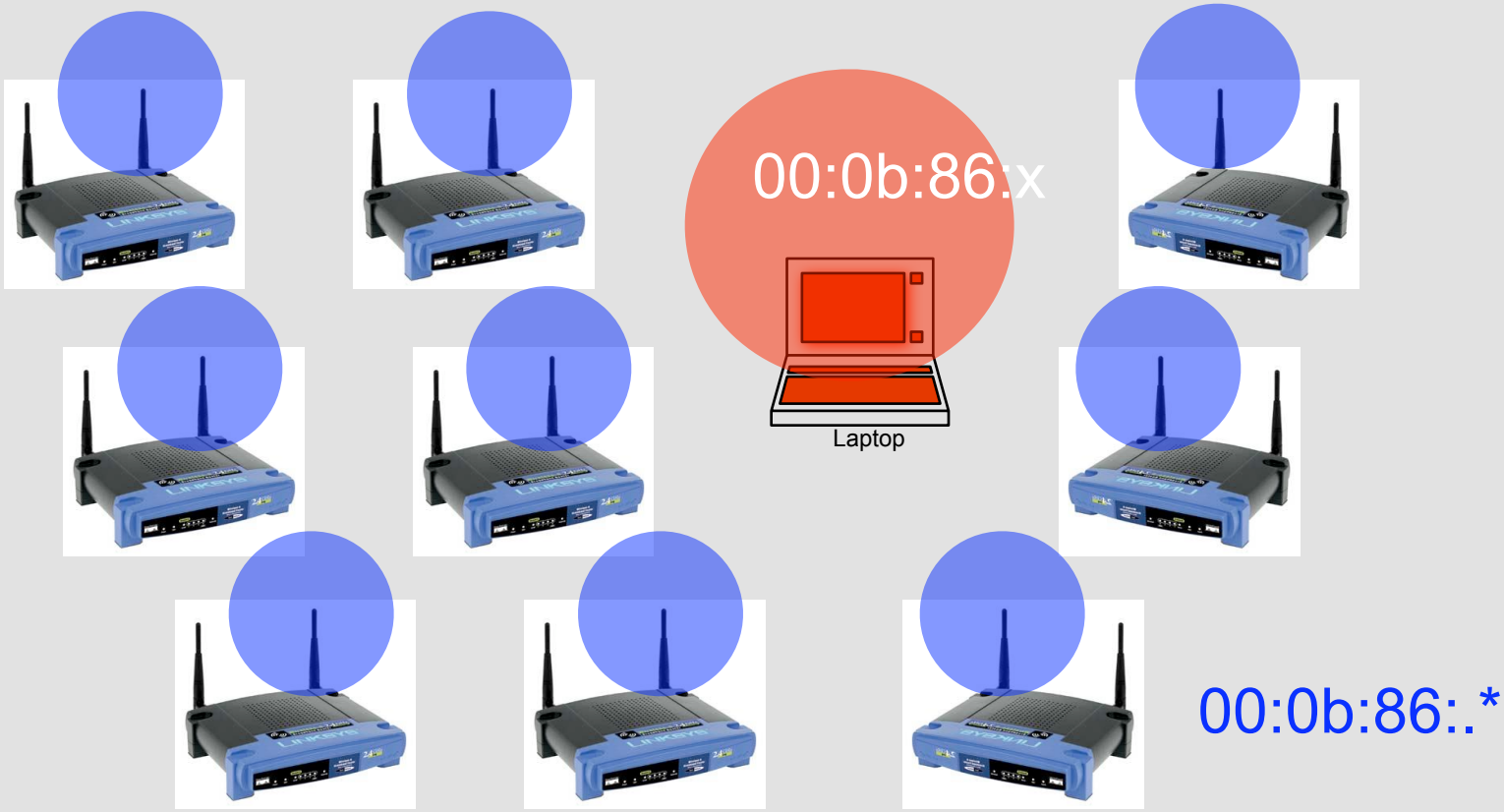
- To perform crypto authentication of an AP, driver must parse complex data structures
- Complex data from untrusted source?
  -- *Is this such a good idea?*

Motivation

Dartmouth College
INSTITUTE FOR SECURITY
TECHNOLOGY STUDIES

Can a client station trust an AP?

Is this AP one of a trusted group, or evil faker?



Motivation

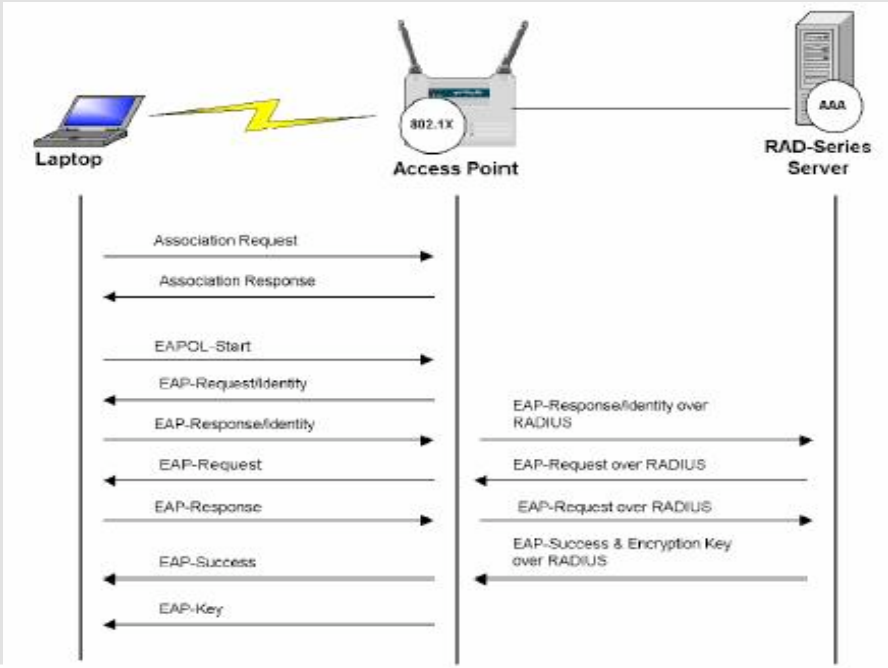
Can a client station trust an AP?

Is this AP one of a trusted group, or evil faker?

Why yes, just exchange some crypto with it, and verify the AP knows the right secrets.

Problem solved, right?

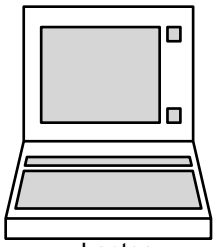
Not exactly: are all these exchanges ***bug-free***?



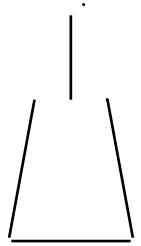
Say it ain't so

7. Application
6. Presentation
5. Session
4. Transport
3. Network
2. Data link
1. Physical

Probe Request -- Probe Response



rates, essid, ...

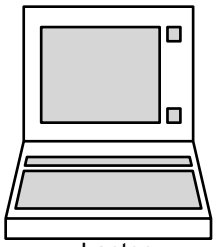


Say it ain't so

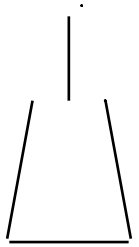
Dartmouth College
INSTITUTE FOR SECURITY
TECHNOLOGY STUDIES

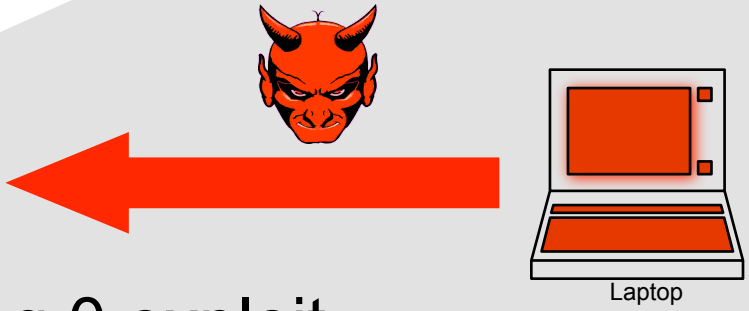
7. Application
6. Presentation
5. Session
4. Transport
3. Network
2. Data link
1. Physical

Probe Request -- Probe Response



rates, essid, ...



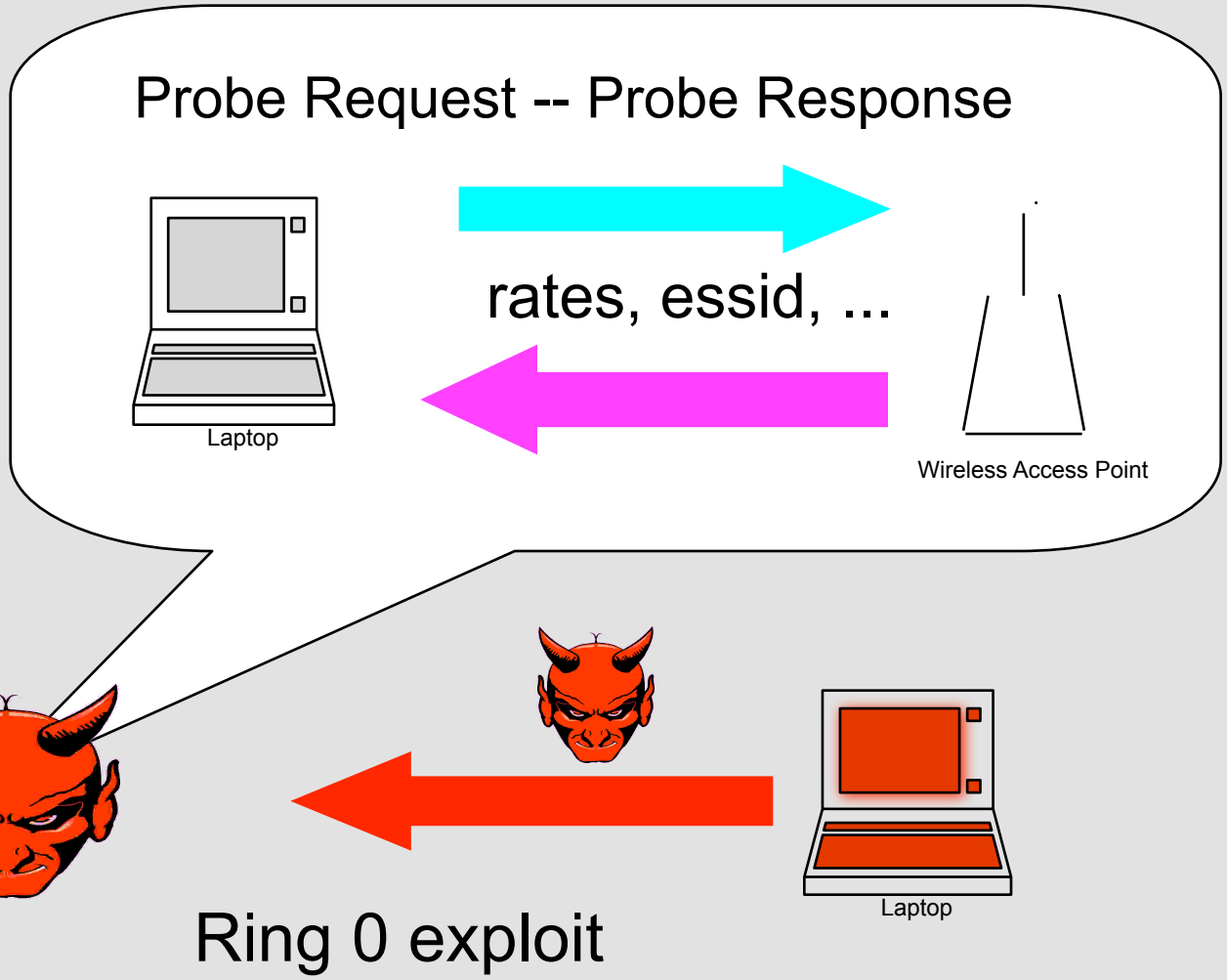


Ring 0 exploit

Say it ain't so

Dartmouth College
INSTITUTE FOR SECURITY
TECHNOLOGY STUDIES

7. Application
6. Presentation
5. Session
4. Transport
3. Network
2. Data link
1. Physical



AP vs. clients

Dartmouth College
INSTITUTE FOR SECURITY TECHNOLOGY STUDIES

Early 802.11: AP = castle,
must fight off barbarians
(unauthorized clients)

Reality: can peasants = clients find the right castle?

- *Dai Zovi, Macaulay:* KARMA ('05)
- *Shmoo:* "Badass tackle..." ('05)
- *Simple Nomad:* "Friendly skies..." ('05)
- *Cache & Maynor:* "Hijackng a MacBook in 60 seconds" ('06)
- Month of kernel bugs (Nov '06)

Fingerprint it!

Fingerprint the AP before trying to authenticate and associate with it:

limit the kinds of accepted data

Must be **simple & cheap** (no RF spectrum analysis, Fourier transforms, etc.)

Follow **IP stack fingerprinting** ideas:

unusual and non-standard header field combinations – but in **link layer** (Layer 2)

TCP/IP fingerprinting

Dartmouth College
INSTITUTE FOR SECURITY
TECHNOLOGY STUDIES

L3, needs an L2 connection

- **Nmap** (1998-2006, ...)
- **Xprobe** (2001, 2005, ...)
- **P0f** (2000, 2006)
- **SinFP** (2005)
- Timing-related: *Ping RTT* (2003), *Clock Skew* (2005)

- Scrubbers: **Norm**, **Bro** (2000-01)
- **Honeyd**, **Morph** (2004-)
- ... ?

L2, not L3

Lots of classic **active** ways to fingerprint TCP/IP stacks:

Nmap, Xprobe, SinFP, Queso, Hping, ...

Timing methods (RTT, clock skew, ...)

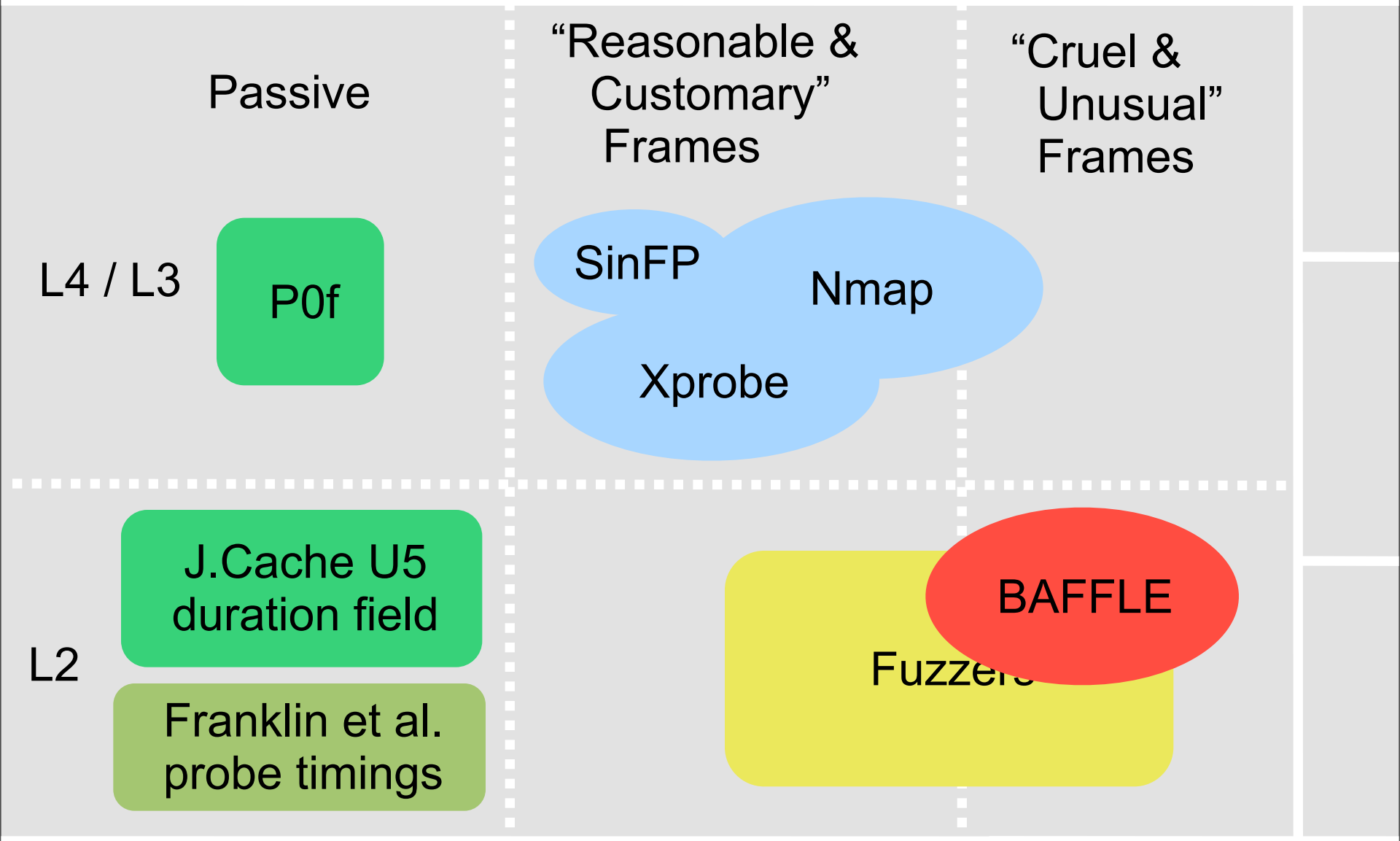
Passive fingerprinting methods:

P0f, 802.11 duration field (*Uninformed.org, Vol 5*)

But: they all need an IP (Layer 3+) connection to be established. We want to **avoid** making it.

Where we fit in

Dartmouth College
INSTITUTE FOR SECURITY TECHNOLOGY STUDIES



BAFFLE

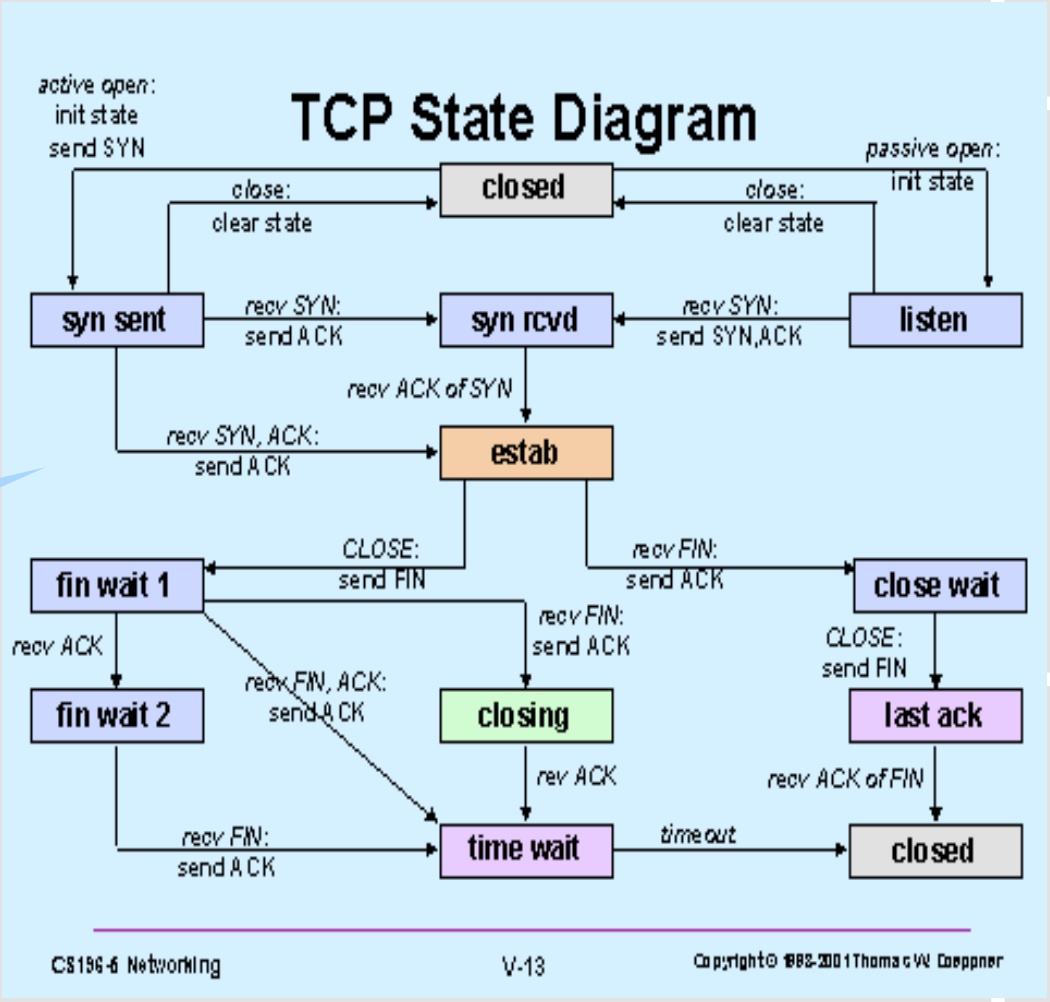
- Written in Ruby 1.8.2
- Ruby LORCON bindings from Metasploit
- Builds Pcap/BPF filters for 802.11 frames from Ruby objects
- Domain-specific language for tests, probes, and for matching responses

Bits and states: TCP

16-bit	32-bit
Source Port	Destination Port
Sequence Number	
Acknowledgement Number (ACK)	
Offset Reserved U A P R S F	Window
Checksum	Urgent Pointer
Options and Padding	

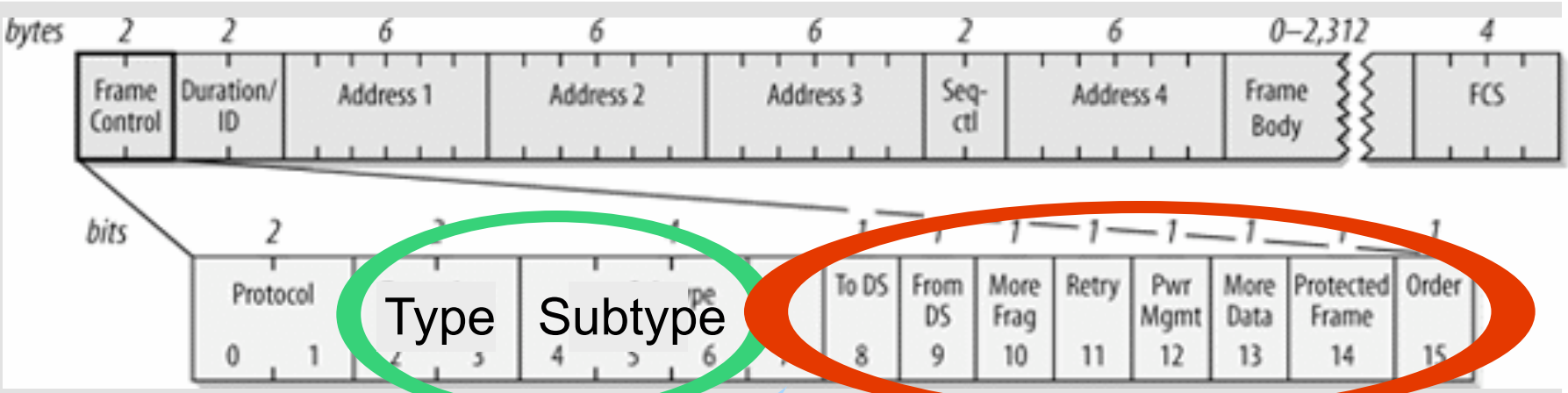
Some fields are meaningless in at least some of the states.

Nmap says hello.



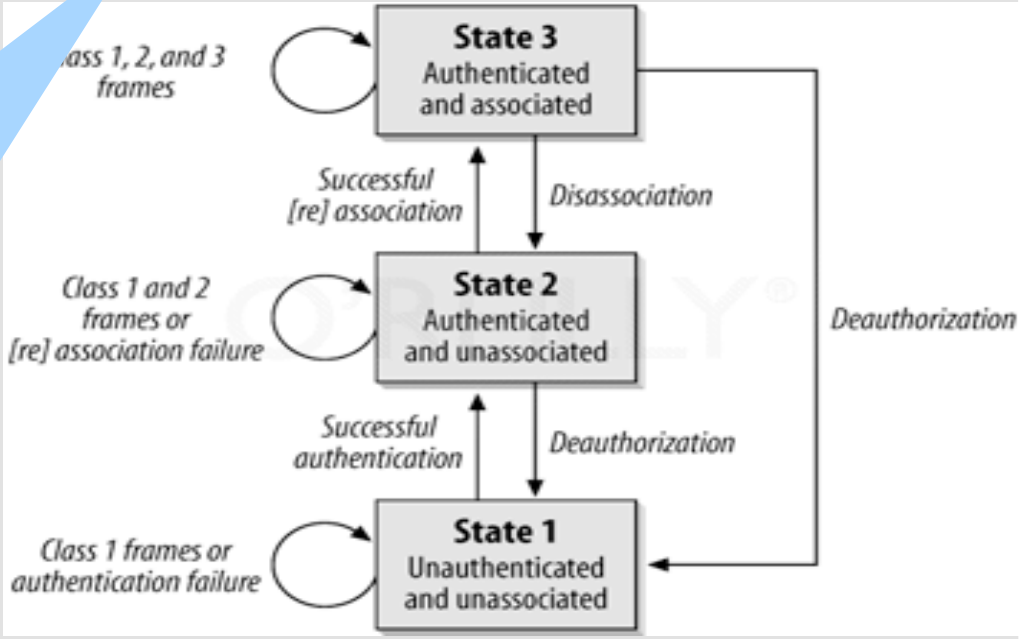
Bits and states: 802.11

Dartmouth College
INSTITUTE FOR SECURITY TECHNOLOGY STUDIES



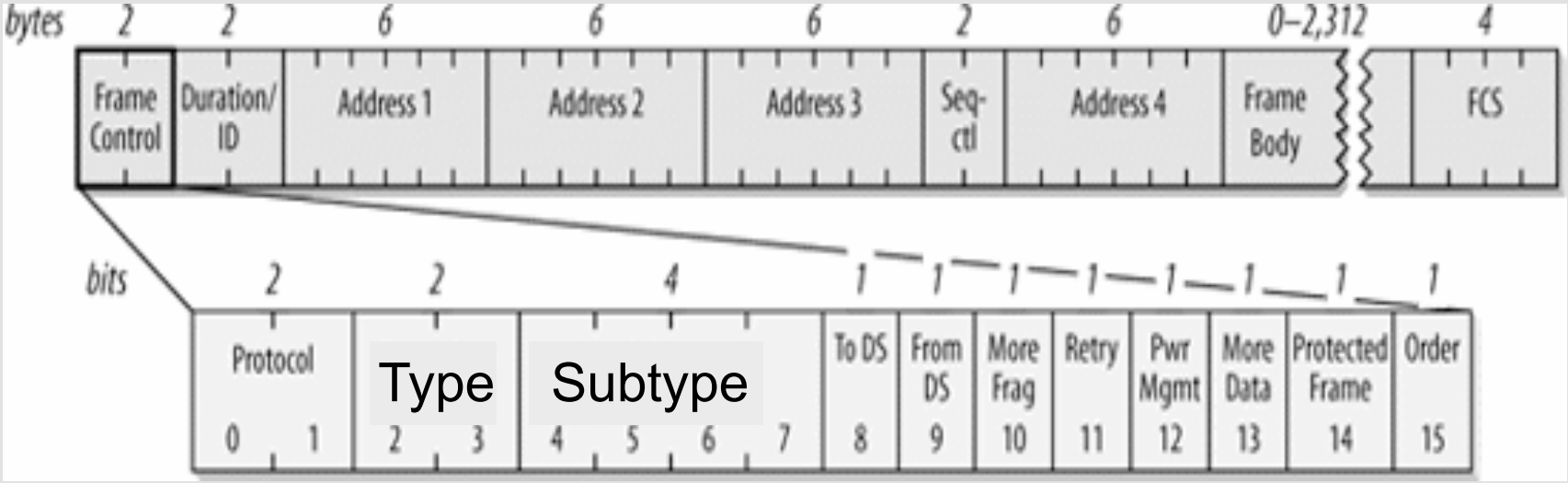
Not all flags make sense for all types & subtypes

Not all flags make sense for all states



802.11 fiddly bits

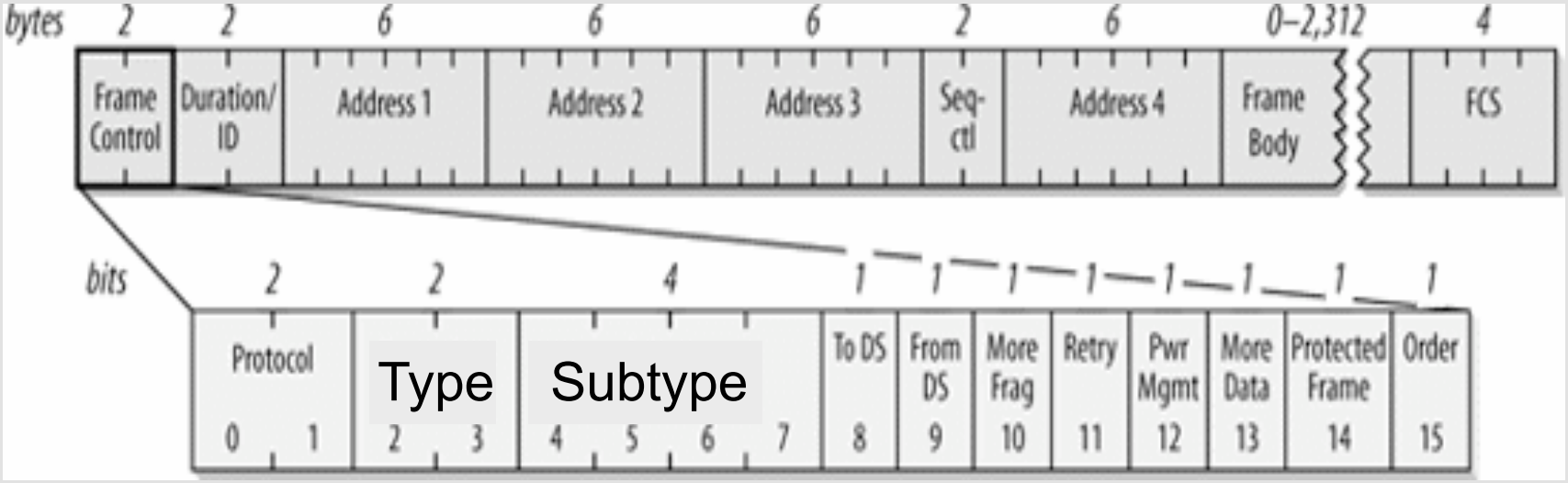
Dartmouth College
INSTITUTE FOR SECURITY TECHNOLOGY STUDIES



Type/Subtype: Mgmt, Control or Data frame type
ToDS, **FromDS**: frame from or to distribution system (zero on management and control frames)
MoreFrag: more L2 fragments to follow
PwrMgmt: station goes into Power Save mode (PS)
MoreData: AP has data buffered for station in Power Save mode

802.11 fiddly bits

Dartmouth College
INSTITUTE FOR SECURITY TECHNOLOGY STUDIES



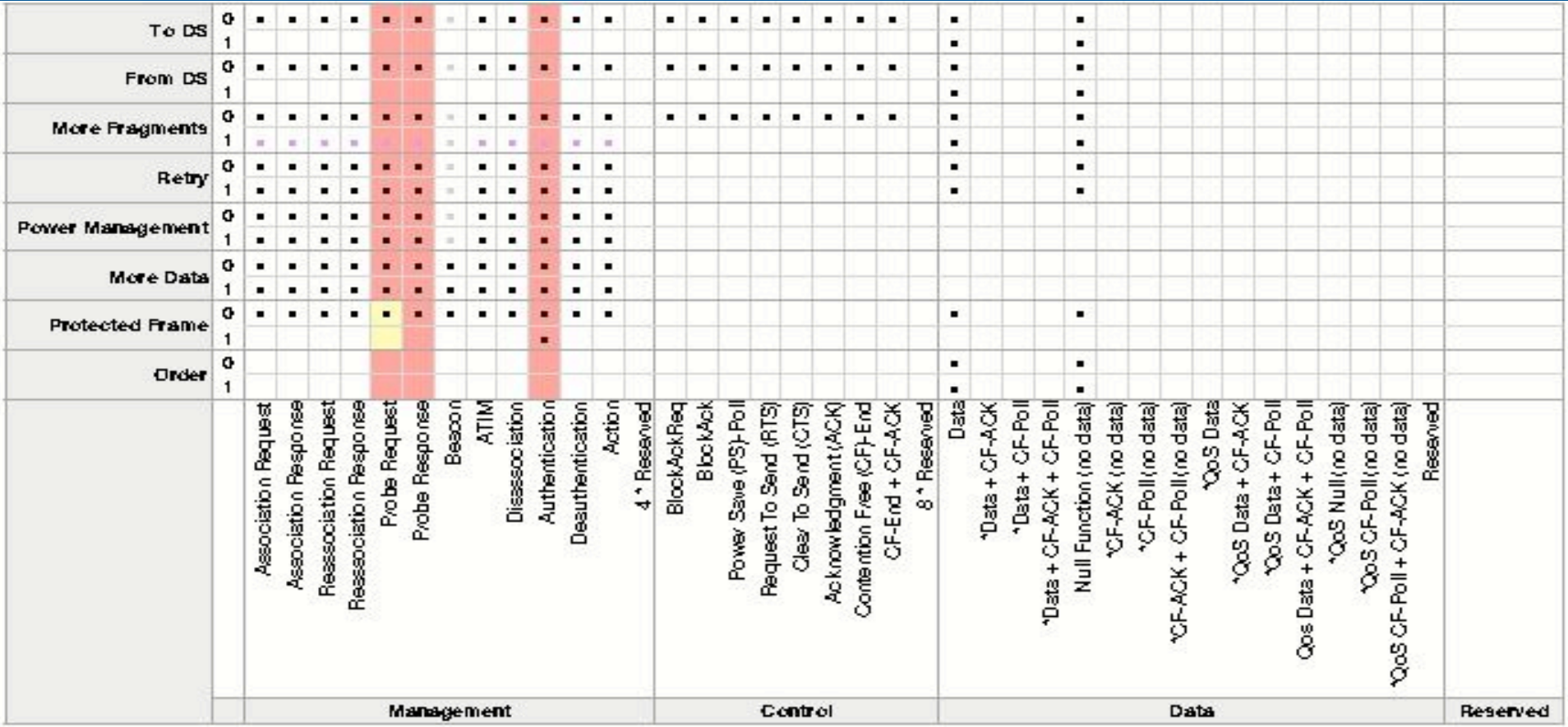
Only **0** makes sense on Mgmt & Ctrl frames

Unusual on Probes

Not for Mgmt frames

So many flags…

Dartmouth College
INSTITUTE FOR SECURITY TECHNOLOGY STUDIES

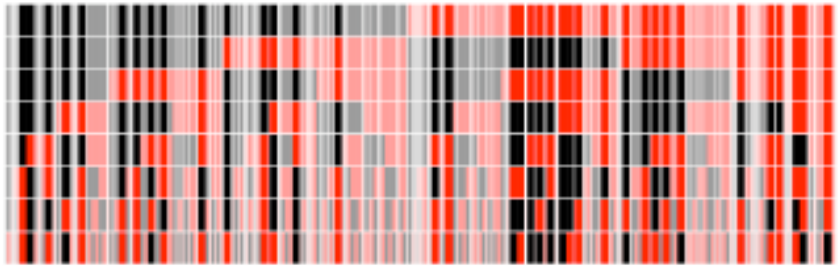


Legend

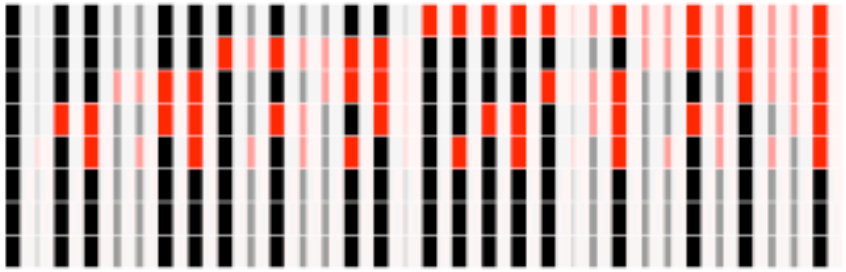
- ▪ Defined by IEEE 802.11 Specification
- ▪ In IEEE 802.11 Specification but purpose seems undefined
- ▪ In IEEE 802.11 Specification but unlikely
- ▪ Tested by BAFFLE
- ▪ Tested by BAFFLE but of limited utility
- □ Not defined in IEEE 802.11 Specification
- * In IEEE 802.11 Specification but mostly unimplemented

Probe Request tests

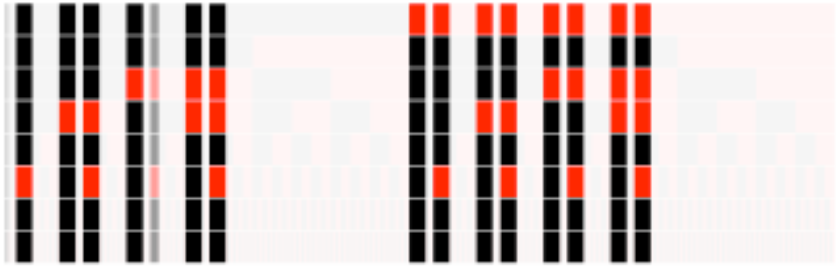
Dartmouth College
INSTITUTE FOR SECURITY
TECHNOLOGY STUDIES



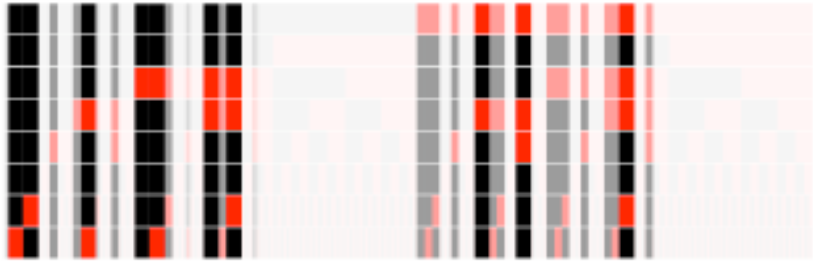
Cisco-Linksys WRT54g *ProbeFCTest*



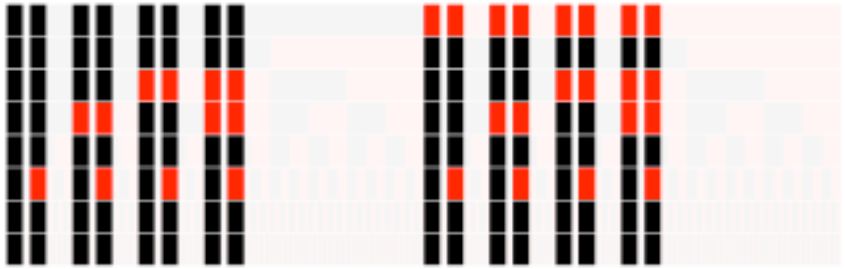
Extrasys WAP-257 *ProbeFCTest*



Madwifi-ng soft AP *ProbeFCTest*



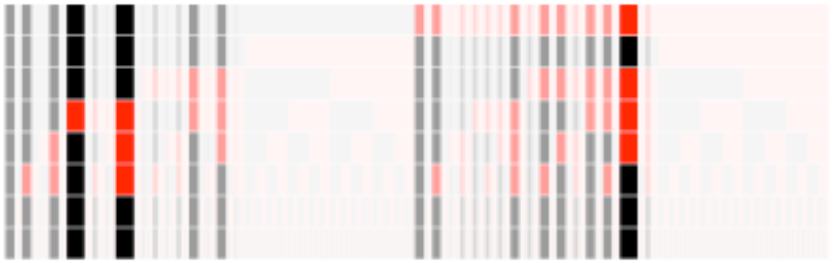
Hostap soft AP *ProbeFCTest*



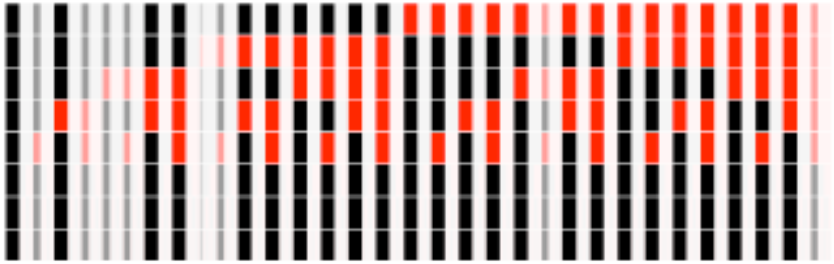
Aruba OpenWRT *ProbeFCTest*

Auth Request tests

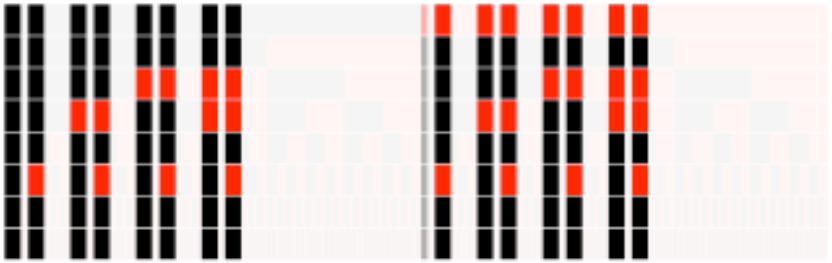
Dartmouth College
INSTITUTE FOR SECURITY
TECHNOLOGY STUDIES



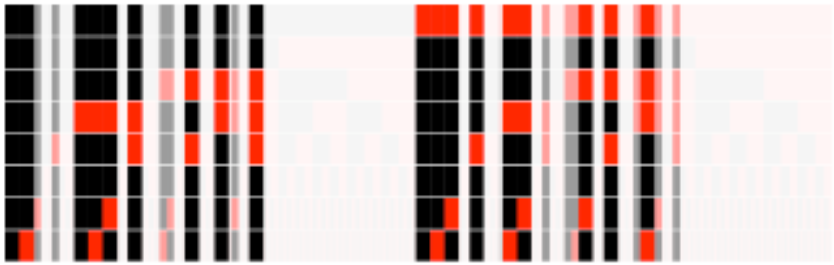
Cisco-Linksys WRT54g *AuthFCTest*



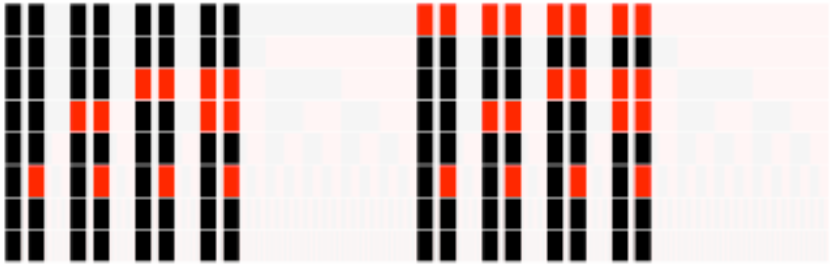
Extrasys WAP-257 *AuthFCTest*



Madwifi-ng soft AP *AuthFCTest*



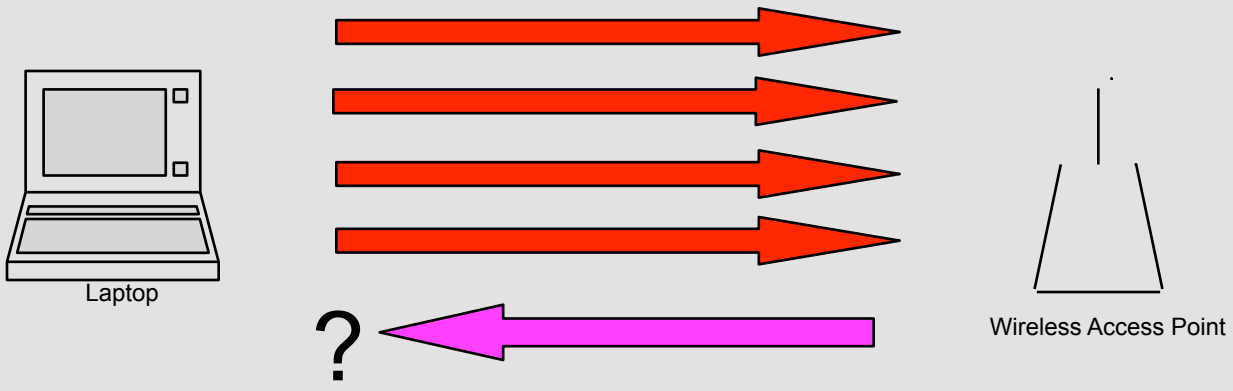
Hostap soft AP *AuthFCTest*



Aruba OpenWRT *AuthFCTest*

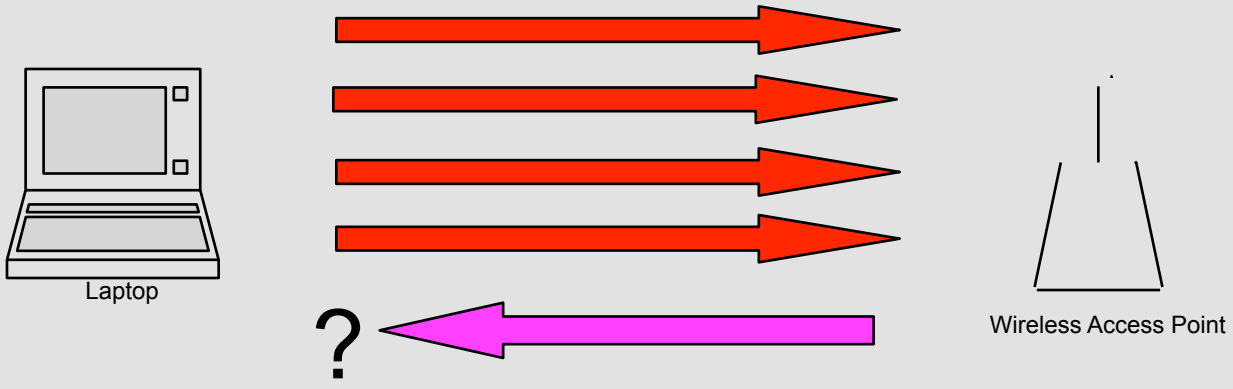
“Secret handshake”

- Send “gibberish” flag combinations in ProbeReq and AuthReq frames
- Watch for reactions (varying MACs helps):
- FromDS, ToDS, MoreFrags, MoreData on STA -> AP frames are all non-standard



“Secret handshake”

Demo



Timing

TCP/IP L3

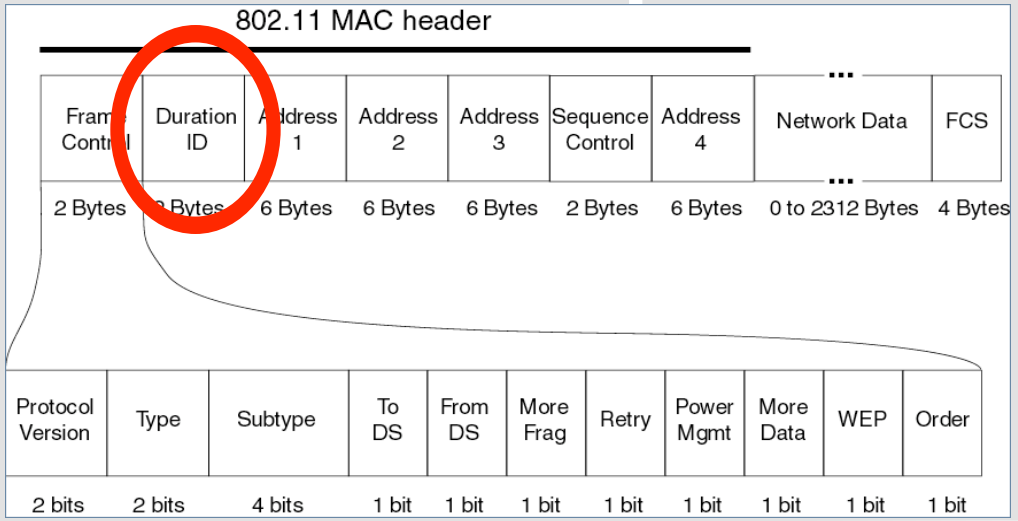
- Tony Capella (DC-11, '03): **Ping RTT**
  "Fashionably late – what your network's RTT tells…"
- Kohno, Broido, Claffy ('05): **Clock Skew**
  "Remote physical device fingerprinting"
- Dan Kaminsky ('05): **IP frag time-outs**

802.11 L2

- Johnny Cache (Uninformed.org 5, '06):
  Statistical analysis of the **duration field**
- Franklin et al (USENIX Sec, '06): **Scanning**
  Time intervals between Probe Req frames

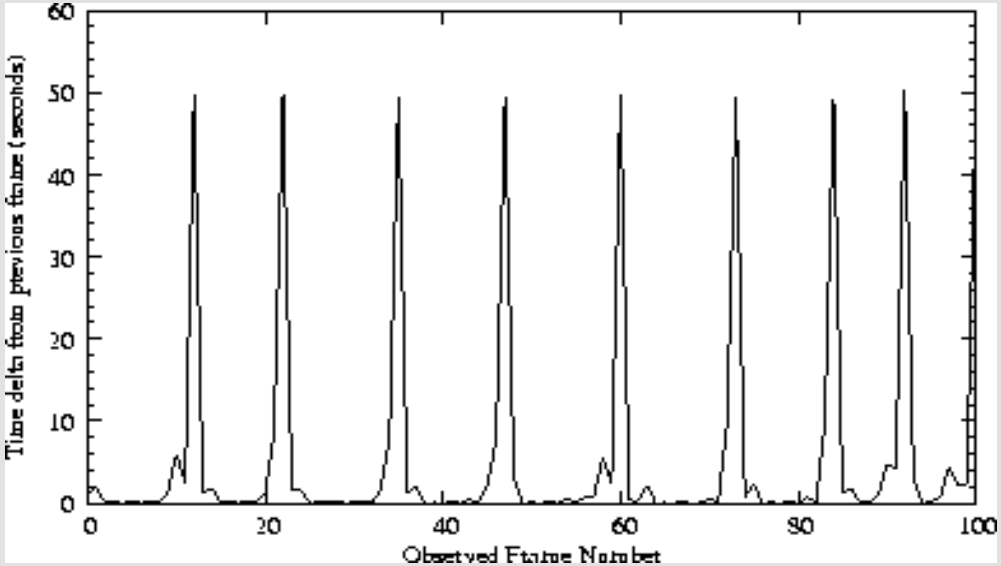
Timing 802.11

- Johnny Cache *(Uninformed, Vol 5):* Statistical analysis of the **duration field** in 802.11 header



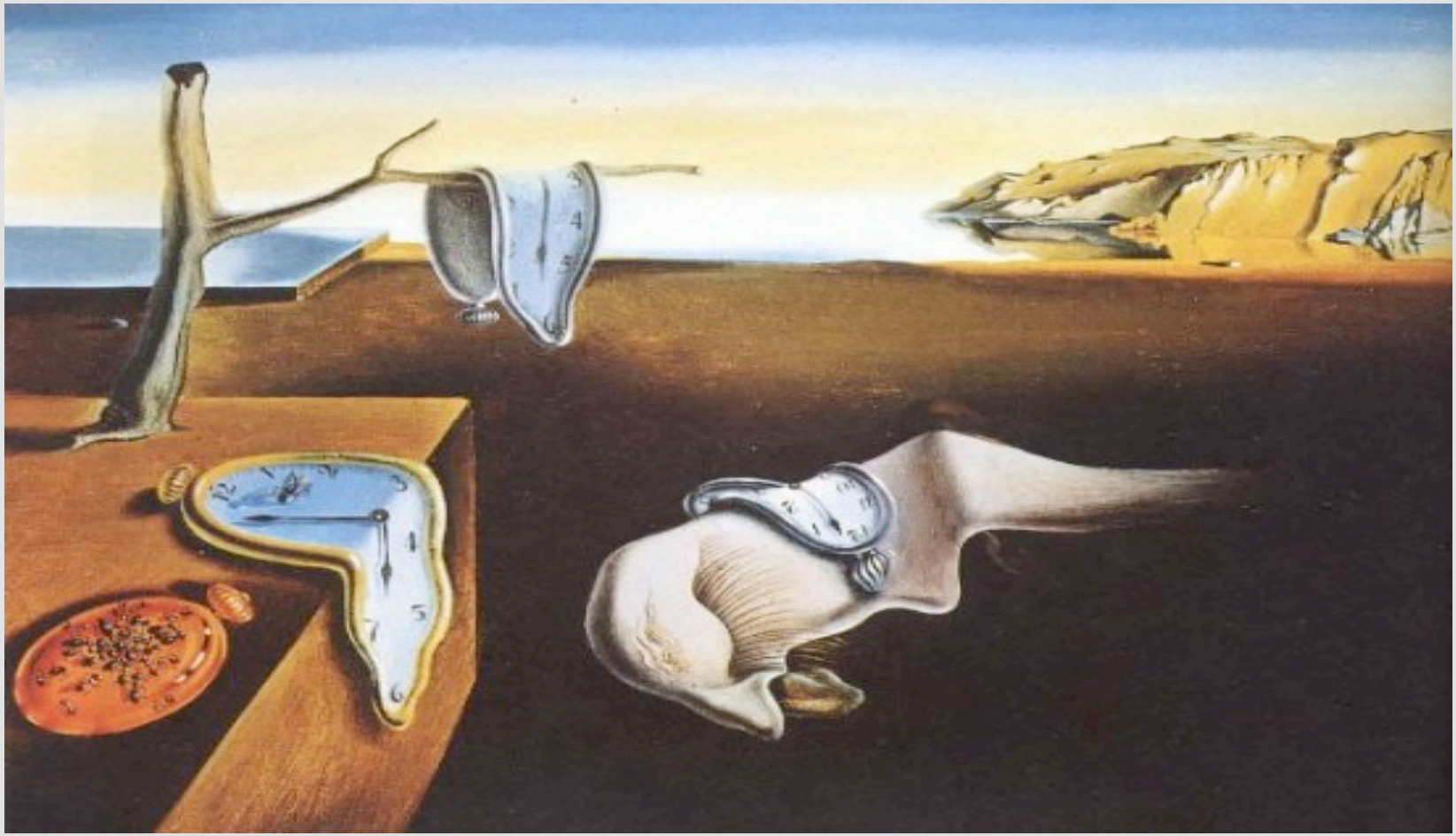
- Franklin et al. *(USENIX Sec, '06):*

Scanning behaviour: intervals between Probe Request frames



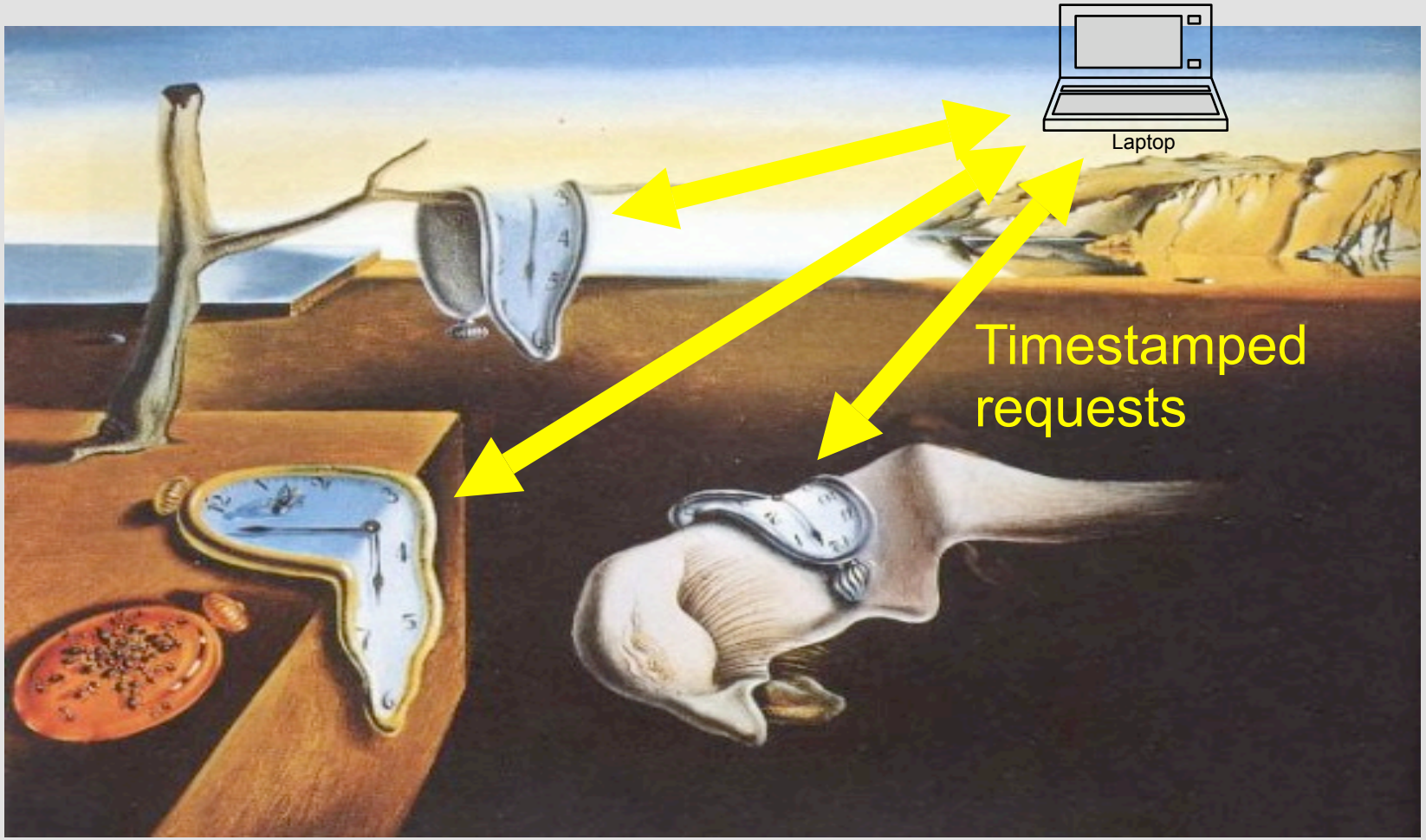
Clock skew

Kohno, Broido, Claffy, *"Remote physical device fingerprinting"*



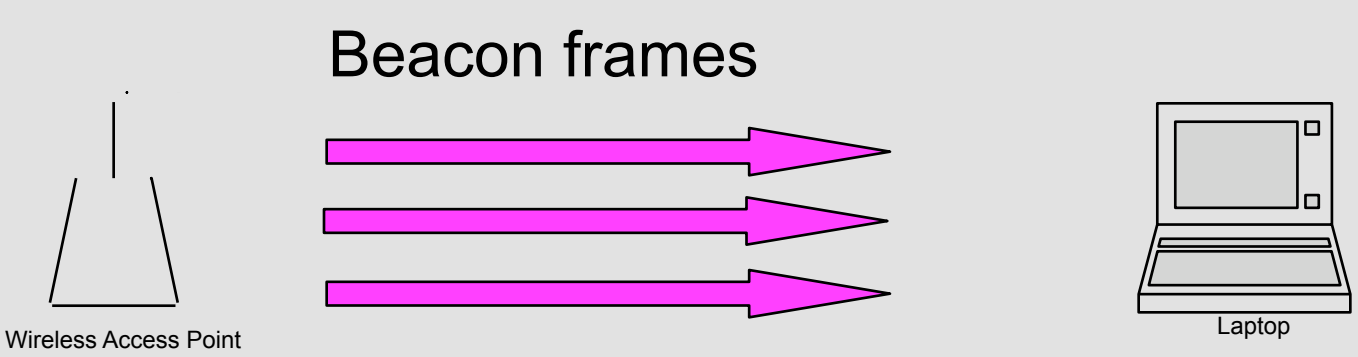
Clock skew

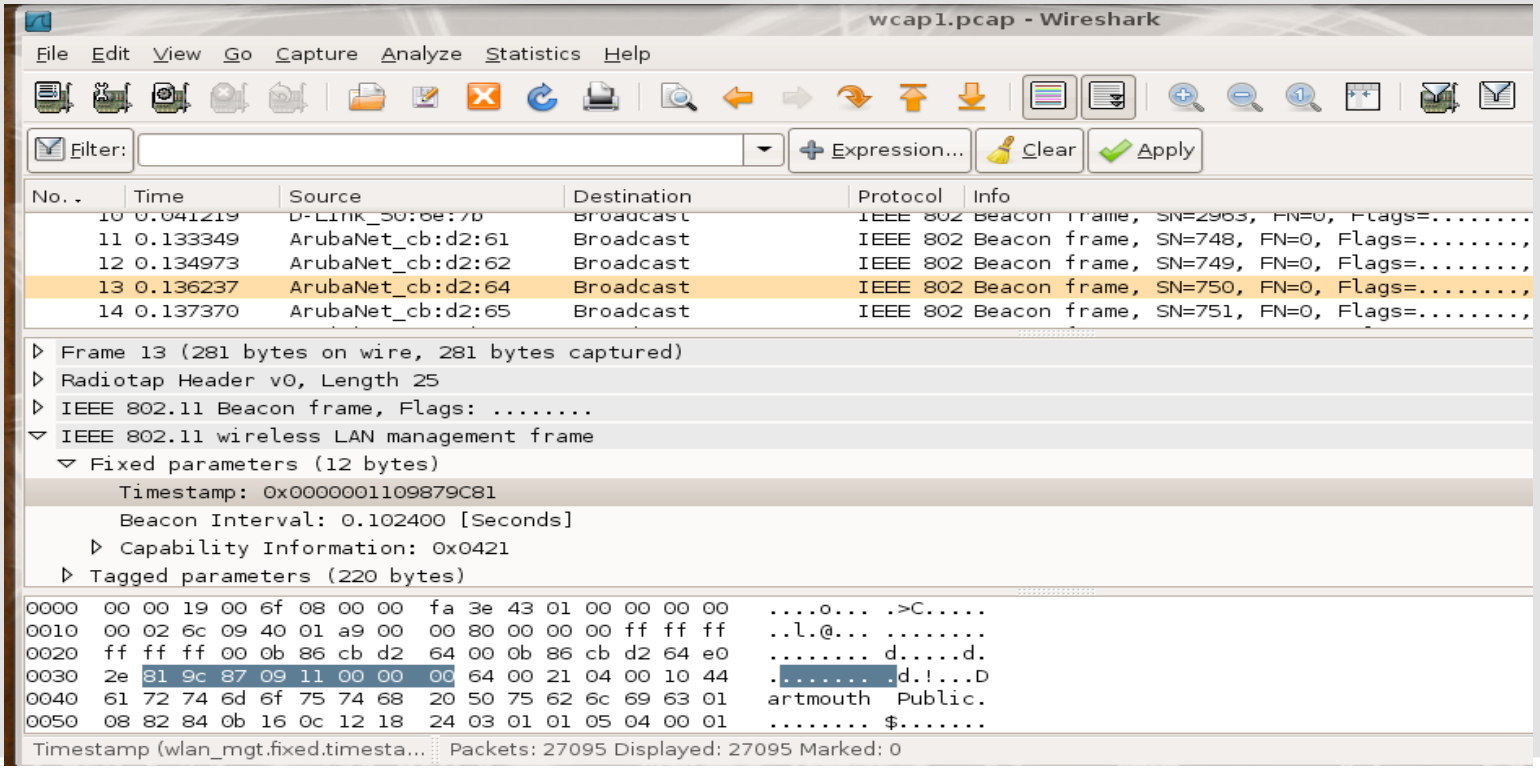
Kohno, Broido, Claffy, *"Remote physical device fingerprinting"*



Beacon clock skew

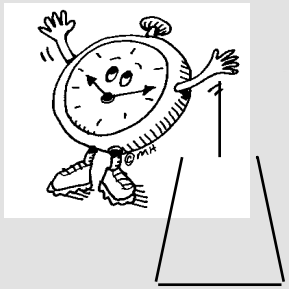
Dartmouth College
INSTITUTE FOR SECURITY TECHNOLOGY STUDIES





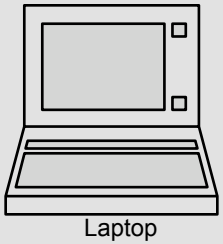
Beacon clock skew

Dartmouth College
INSTITUTE FOR SECURITY TECHNOLOGY STUDIES

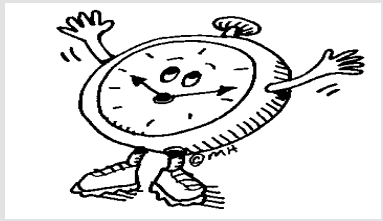


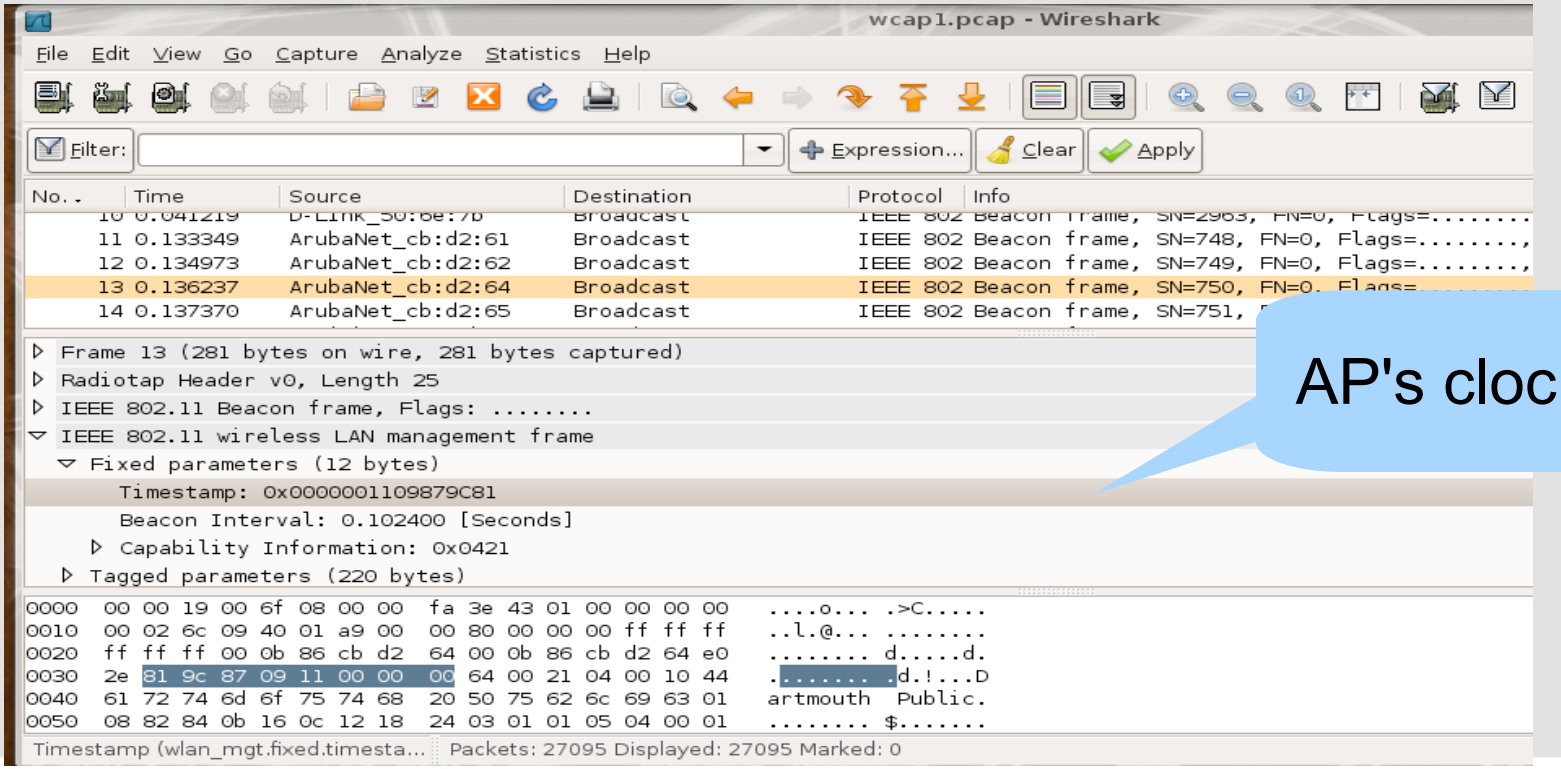
Wireless Access Point

Beacon frames



Laptop

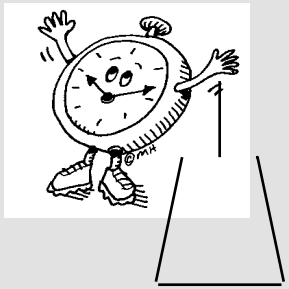




AP's clock time

Beacon clock skew

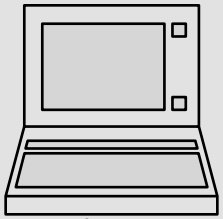
Dartmouth College
INSTITUTE FOR SECURITY
TECHNOLOGY STUDIES



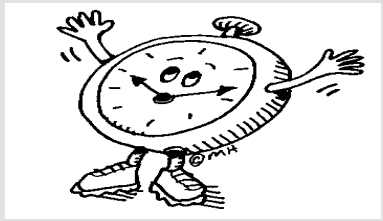
Wireless Access Point

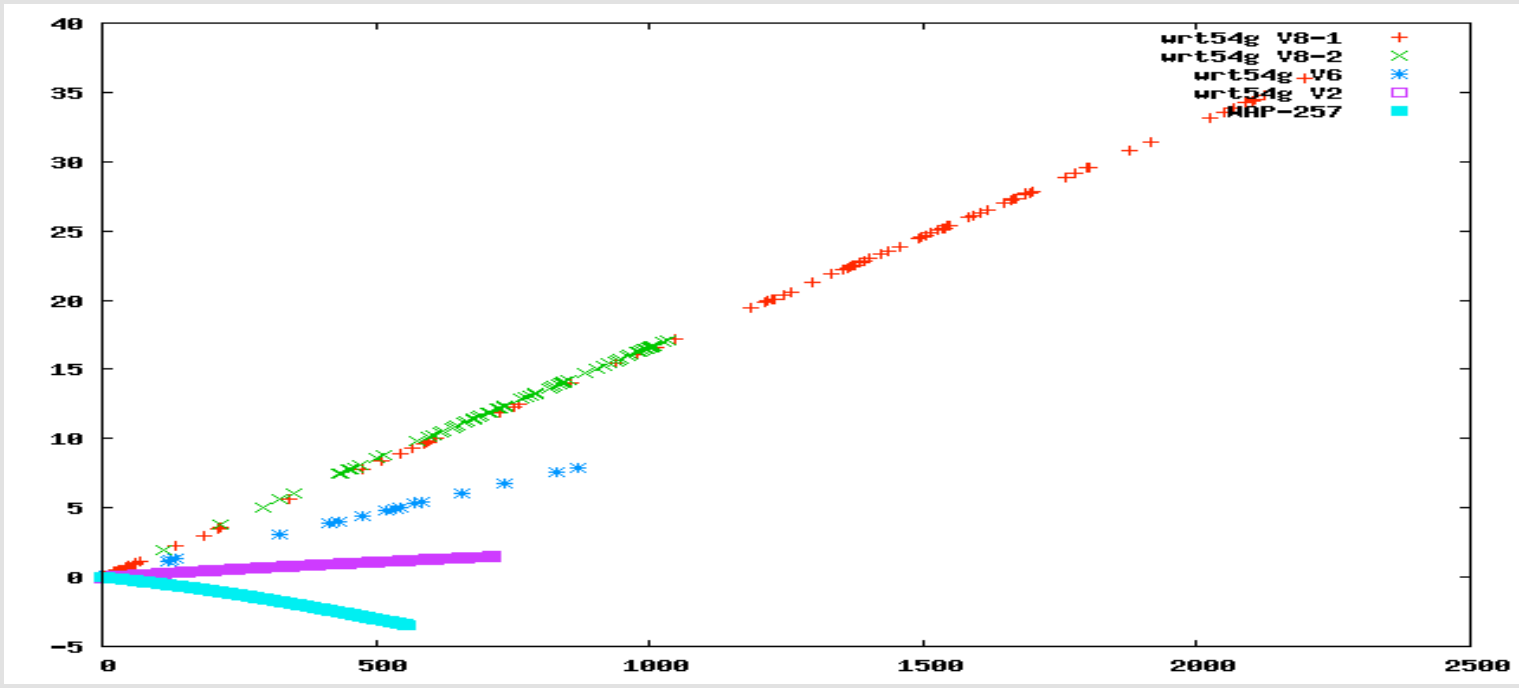
AP

Beacon frames



Laptop





sen-
sor

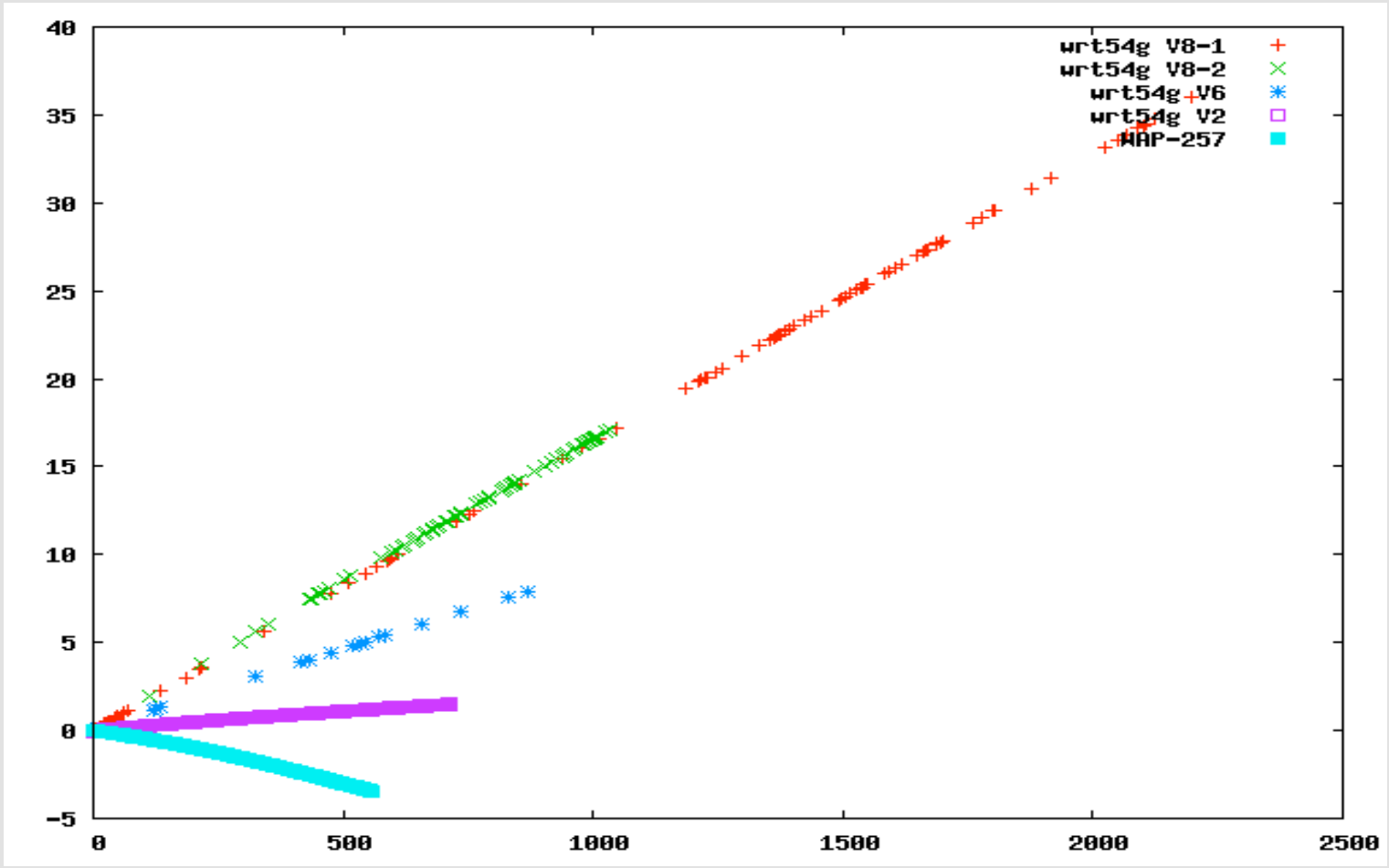
AP beacon clock skew

Dartmouth College
INSTITUTE FOR SECURITY
TECHNOLOGY STUDIES

- Beacon frames contain AP clock's timestamp
- Each HW clock drift differently; **skew** is the derivative of the clock's offsets against another clock *(cf. Kohno, Broido, Claffy '05)*
- Issues:
  - AP clock's unique skew can be estimated reliably within 1-2 mins
  - Similar AP models have closer skews
  - Faking (e.g., with a laptop + Wi-Fi card in master mode) is hard enough

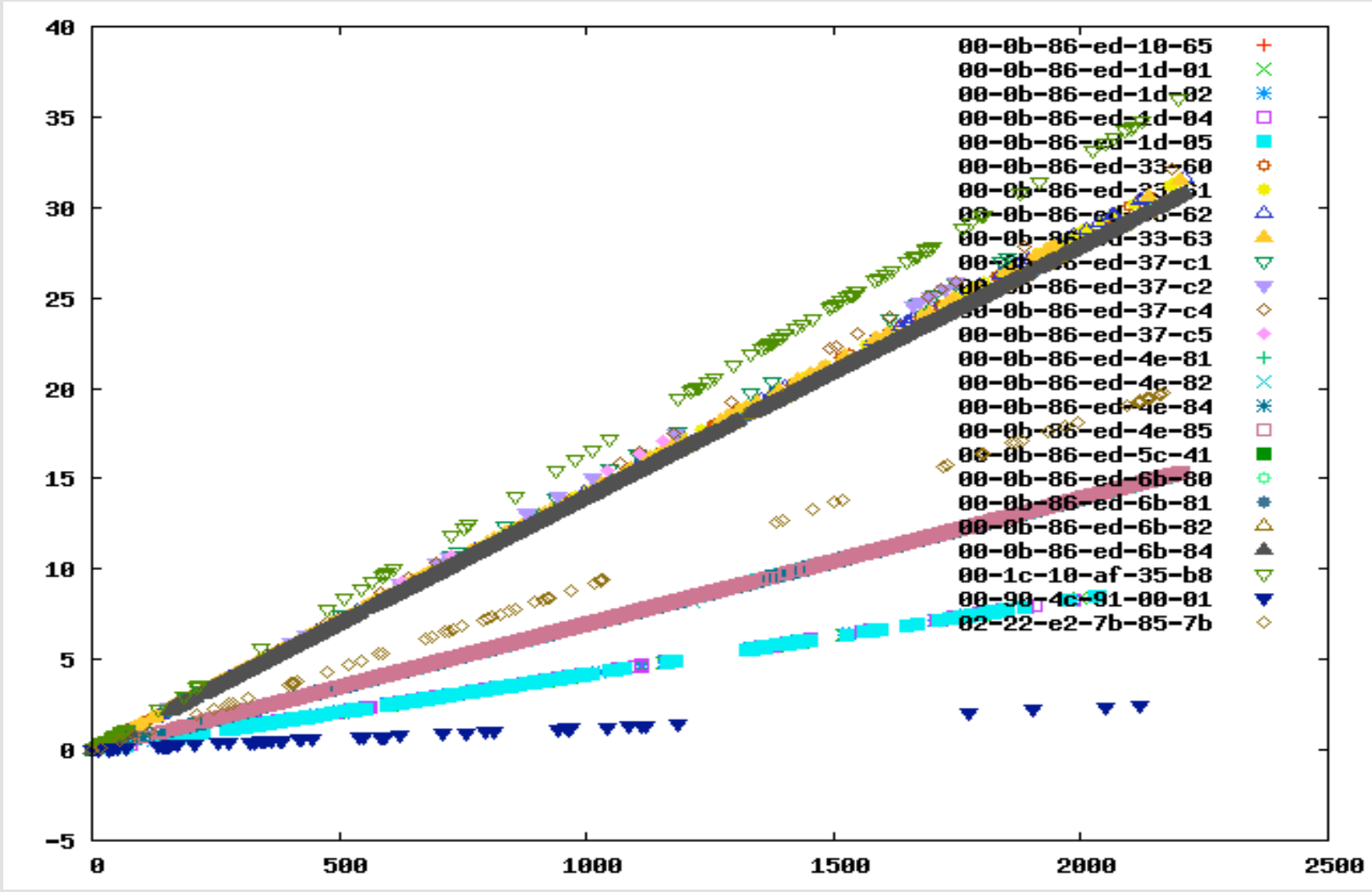
AP beacon clock skew

Dartmouth College
INSTITUTE FOR SECURITY
TECHNOLOGY STUDIES



AP beacon clock skew

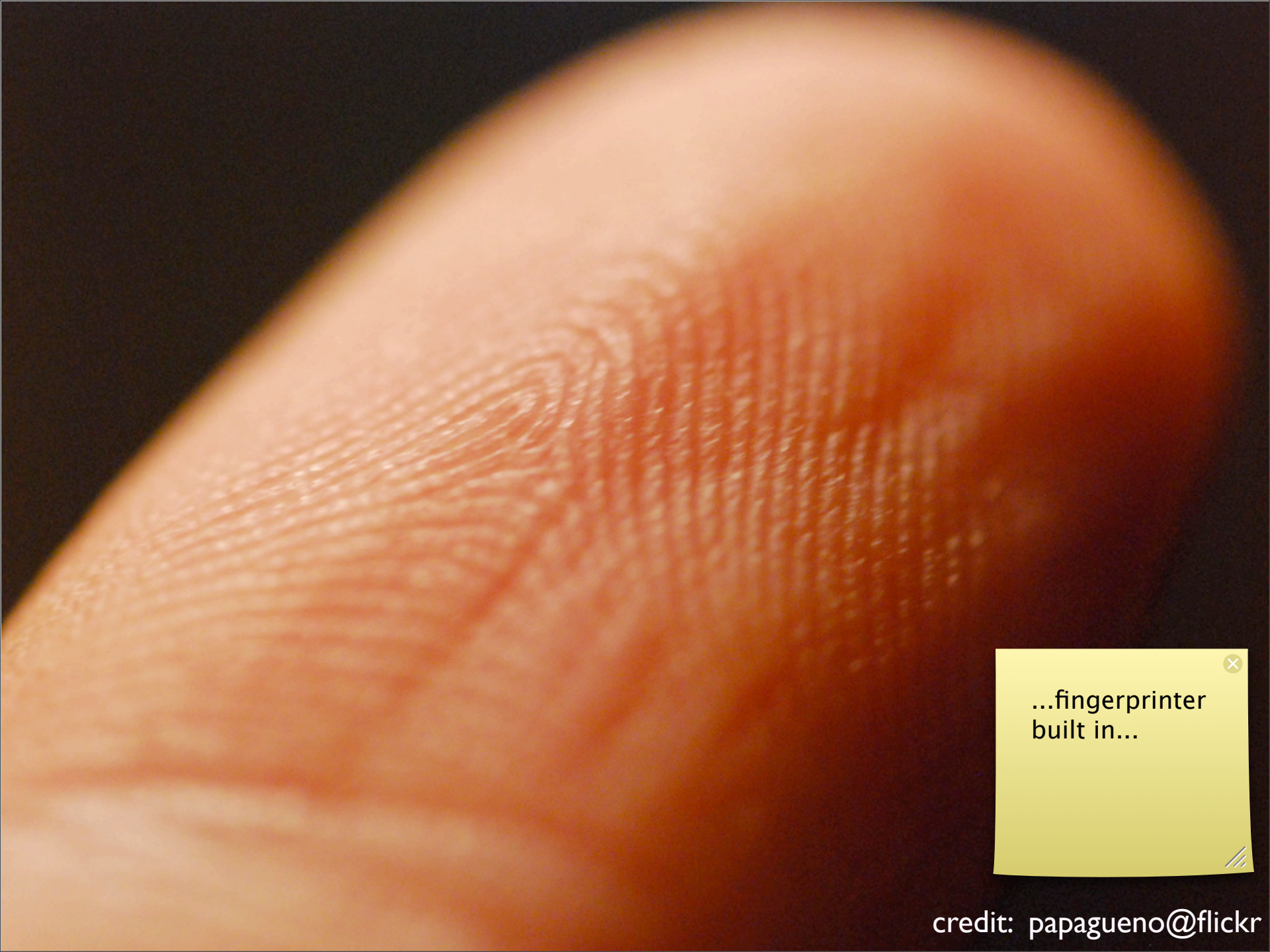
Dartmouth College
INSTITUTE FOR SECURITY
TECHNOLOGY STUDIES



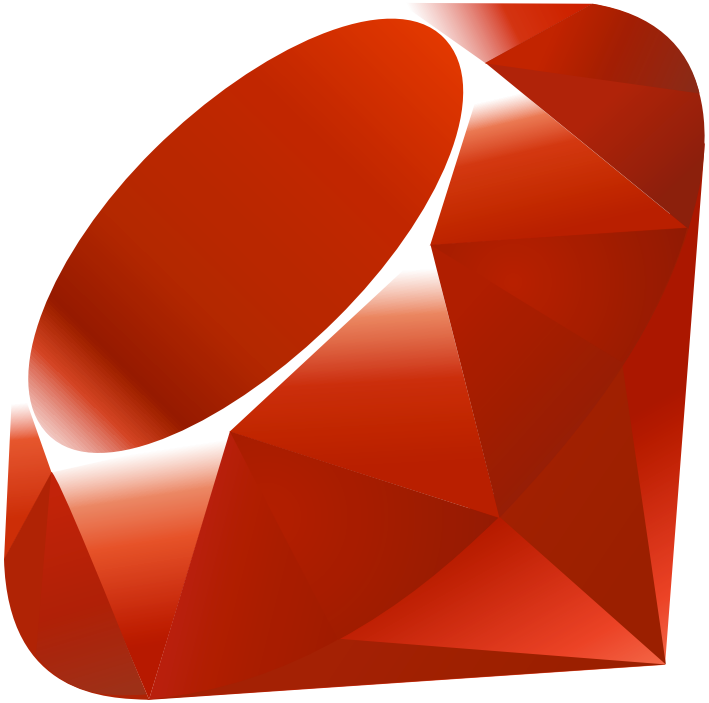
baffle architecture

I am going to talk briefly about the architecture of Baffle. But first, to recap what Baffle is. Baffle is a...

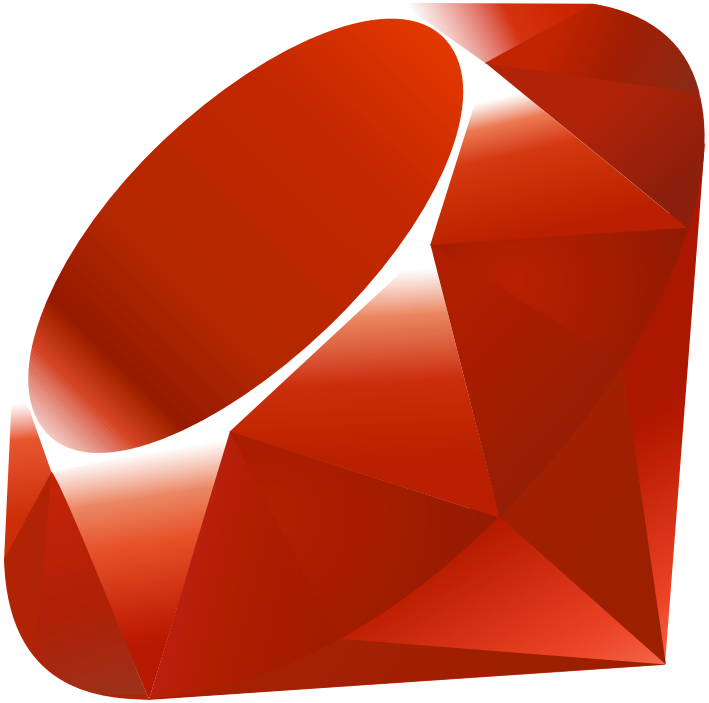
...layer 2 wireless (that is the 802.11 mac layer)...



credit: papagueno@flickr



...ruby and...



+ C

...C. The primary C components we utilize are...

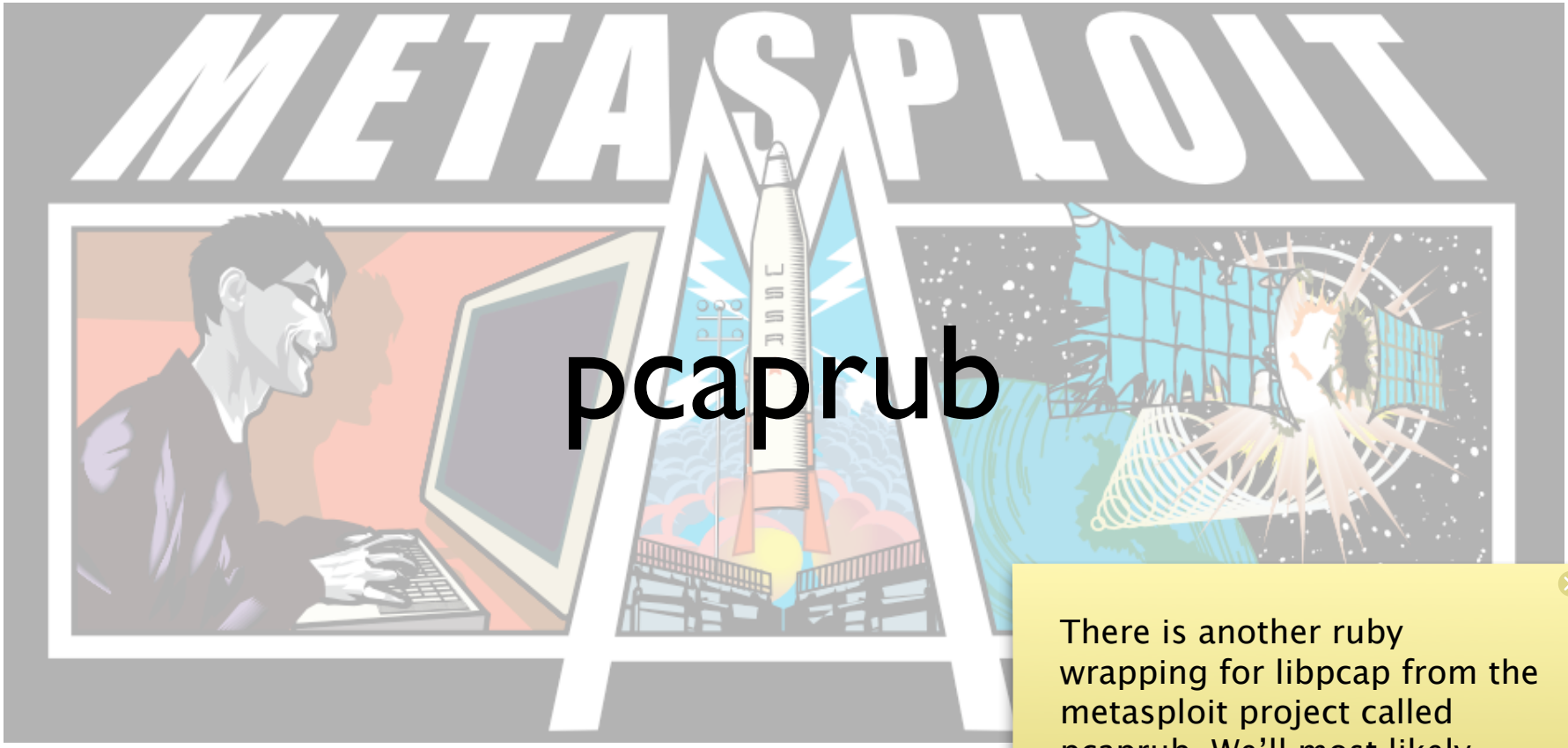
libpcap

...libpcap which allows us to capture packets, of course. We have a ruby wrapper around this for libpcap called...

rb-pcap

...rb-pcap. This is a very basic wrapper and was really created just to get the job done.

```ruby
Capture.open(:device => 'ath0') do |capture|
  capture.each do |packet|
    # ...process packet...
  end
end
```

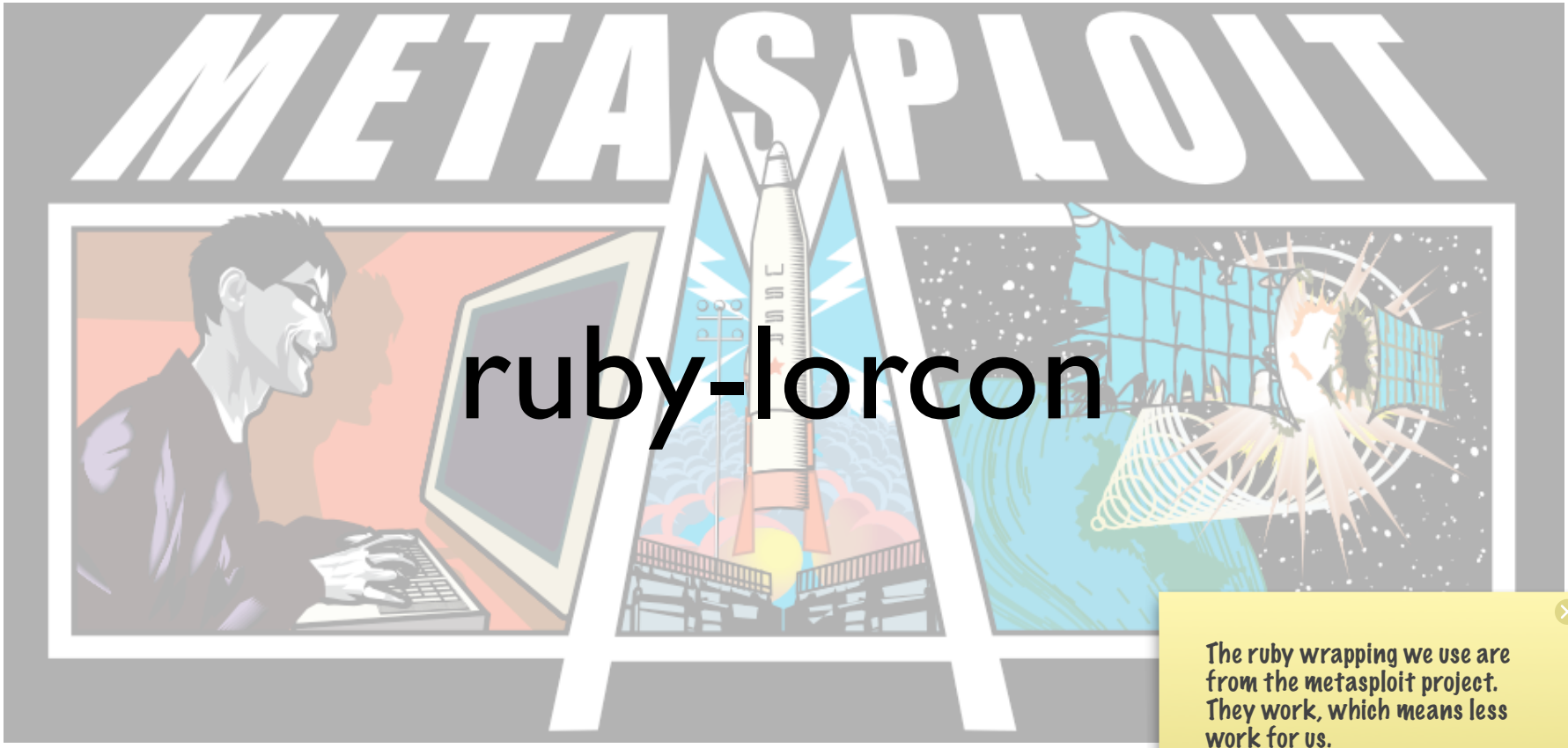


pcaprub

There is another ruby wrapping for libpcap from the metasploit project called pcaprub. We'll most likely start using pcaprub since its not cool to reinvent the wheel.

lorcon

On the injection side we use lorcon. Lorcon stands for Loss Of Radio Connectivity. For those of you who don't know what lorcon is, it is a generic library for injecting 802.11 frames. It's a beautiful abstraction layer that makes our life easy.



ruby-lorcon

The ruby wrapping we use are from the metasploit project. They work, which means less work for us.

```
device = Lorcon::Device.new('ath0', 'madwifing')
device.fmode = 'INJMON'
device.channel = 11

packet = '...'

device.write packet, 1, 0
```

you specify the interface & driver. set the mode and channel. then you can write packets. the last two parameters are the count and delay.

dot11

for parsing and creating 802.11 packets, we created something called dot11. Here is the syntax...

```
Dot11.new(:subtype => 0xb,
          :addr1 => 'ba:aa:ad:f0:00:0d',
          :addr2 => 'ba:aa:ad:be:ee:ef',
          :addr3 => 'ba:aa:ad:f0:00:0d') /
Dot11Auth.new(:seqnum => 1)
```

If this looks similar to scapy, there is a reason. We debated using python or ruby and it ultimately came down to who knows what best. We already had some experience with Ruby so we went with it. That said, we liked the scapy syntax so we tried to mimic it as much as possible in Ruby.

scapy / scruby

There is of course something called scruby now that metasploit uses in version 3.1.

```
Dot11(subtype=0xb,
      addr1='ba:aa:ad:f0:00:0d',
      addr2='ba:aa:ad:be:ee:ef',
      addr3='ba:aa:ad:f0:00:0d') / Dot11Auth(seqnum=0)
```

why did we write dot11?

We were unaware of the scruby project when we first started and when we became aware of it there were some things we didn't like about it. Namely, it didn't have support for the Dot11 related classes that were already present in scapy. We stuck with what we had instead of converting but now would probably be a good time to switch to scruby since it has support for Dot11 as of version 0.3. But there are some features of our implementation that we would need to port over first.

```
Dot11.new(:type => 0, :subtype => 0xb).to_filter


=> ((ether[0:1] & 0xc) >> 2 = 0x0) &&
   ((ether[0:1] & 0xf0) >> 4 = 0xb)
```

```
Dot11.new(:type => 0...4, :subtype => 0...16).to_filter


=> (((ether[0:1] & 0xc) >> 2 >= 0x0 &&
     (ether[0:1] & 0xc) >> 2 < 0x4)) &&
   (((ether[0:1] & 0xf0) >> 4 >= 0x0 &&
     (ether[0:1] & 0xf0) >> 4 < 0x10))
```

```ruby
Dot11.new(:type => 0...4,
          :subtype => 0...16).each do |packet|
  # ... do something with packet ...
end
```

kind of similar to the fuzz function present in scapy except that we want to fuzz specific fields only and leave everything else default

how do we fingerprint?

so i talked about the primary components that make of Baffle but how do we utilize these components to fingerprint access points? It's quite

sniff, inject & compute

we sniff, inject & compute. sniffing and injection take place on their own threads. once those are complete we compute a vector and determine how similar that vector is to other vectors in our database using some fancy math. And all this logic in contained in what we call a probe...


02895-100
LOT# 492006
MANUFACTURED BY
Welch Allyn
CE 0051
PROBE WELL KIT OEM RECTAL 9'
7420 Carroll Road
San Diego, CA 92121
okay, this is an anal probe. i don't think our probes are all that painful but i wouldn't really know. our probes...
credit: juhansonin@flickr

```
probe "authreq_flags" do
  inject(0..255) do |options, flags|
    local_bssid   = "ba:ad:"
    local_bssid  << options.bssid.slice(-5..-1)
    local_bssid  << ":00:"
    local_bssid  << "00#{flags.to_s(16)}".slice(-2..-1)

    Dot11::Dot11.new(:subtype =>   0xb,
                     :flags =>     flags,
                     :addr1 =>     options.bssid,
                     :addr2 =>     local_bssid,
                     :addr3 =>     options.bssid,
                     :payload =>   Dot11::Dot11::Dot11Auth.new(:seqnum => 0x0001))
  end

  filter :subtype_addr1 do |options|
    local_bssid   = "ba:ad:"
    local_bssid  << options.bssid.slice(-5..-1)
    local_bssid  << ":00:00/32"

    Dot11::Dot11.new(:type => 0, :subtype => 0xb, :addr1 => local_bssid)
  end

  capture :subtype_addr1 do |packet|
    packet.addr1.to_i & 0xff
  end

  compute_vector do |samples|
    vector = Array.new(256, 0)

    samples.each do |sample|
      sample.each do |flags|
        vector[flags] += 1
      end
    end

    vector
  end
end
```

look like this. basically this is a nice little domain specific language we created to capture the operations necessary to fingerprint. you

```
probe "authreq_flags" do
  inject(0..255) do |options, flags|
    ...
  end

  filter :subtype_addr1 do |options|
    ...
  end

  capture :subtype_addr1 do |packet|
    ...
  end

  compute_vector do |samples|
    ...
  end
end
```

probe has:
* name
* inject logic
* filter & capture logic
* compute logic

there are also some other things you can specify such as the number of times you want to inject the packets.

```ruby
inject(0..255) do |options, flags|
  local_bssid   = "ba:ad:"
  local_bssid  << options.bssid.slice(-5..-1)
  local_bssid  << ":00:"
  local_bssid  << "00#{flags.to_s(16)}".slice(-2..-1)

  Dot11::Dot11.new(:subtype =>   0xb,
                   :flags =>     flags,
                   :addr1 =>     options.bssid,
                   :addr2 =>     local_bssid,
                   :addr3 =>     options.bssid,
                   :payload =>   Dot11::Dot11::Dot11Auth.new(:seqnum => 0x0001))
end
```

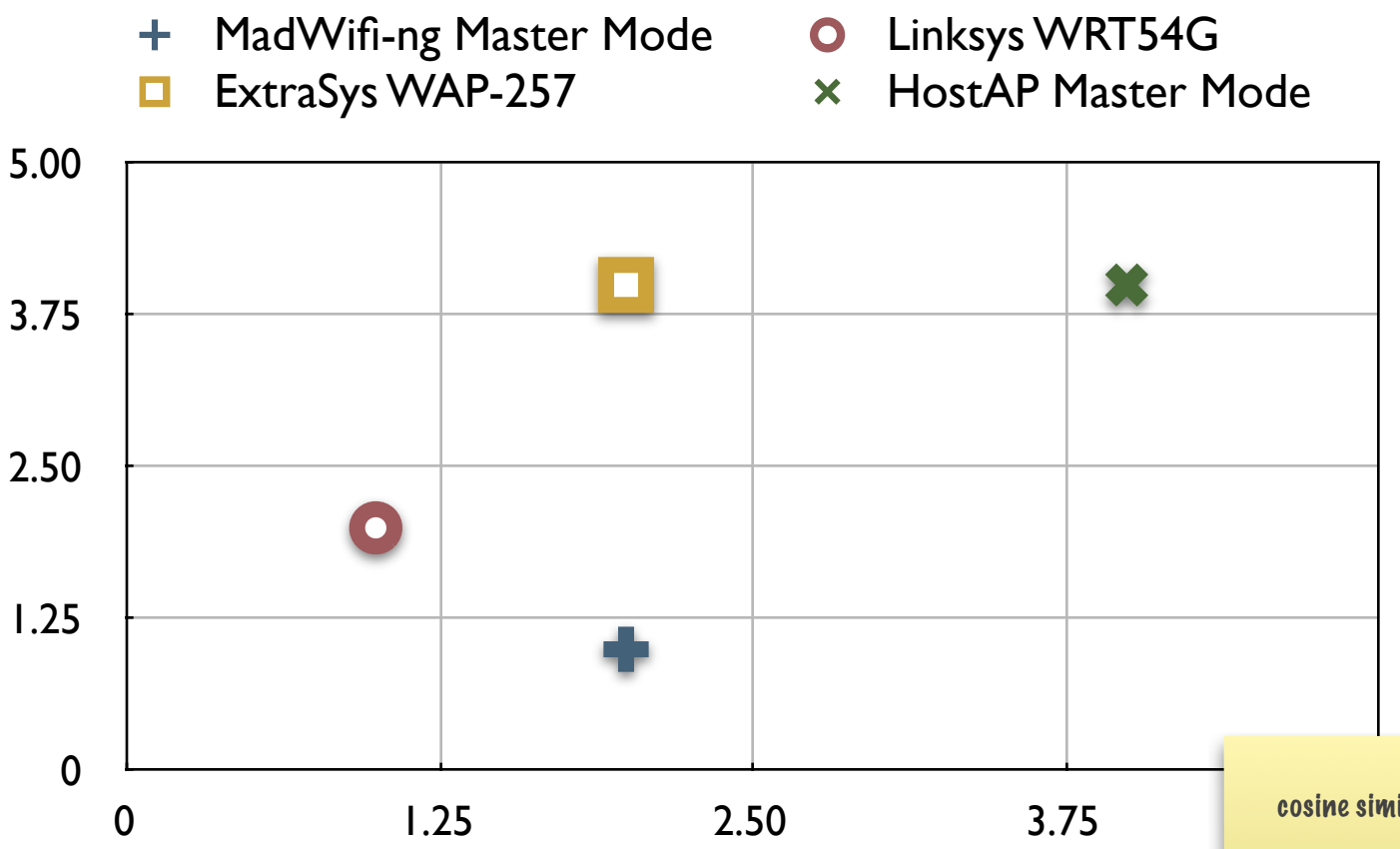
cartesian product of arguments given to inject

explain local_bssid and reasoning behind construction

```
filter :subtype_addr1 do |options|
  local_bssid   = "ba:ad:"
  local_bssid  << options.bssid.slice(-5..-1)
  local_bssid  << ":00:00/32"

  Dot11::Dot11.new(:type => 0, :subtype => 0xb, :addr1 => local_bssid)
end


capture :subtype_addr1 do |packet|
  packet.addr1.to_i & 0xff
end
```

```
compute_vector do |samples|
  vector = Array.new(256, 0)

  samples.each do |sample|
    sample.each do |flags|
      vector[flags] += 1
    end
  end

  vector
end
```

singular value decomposition



cosine similarity

computing the angle between these two vectors

Source & Thanks

http://baffle.cs.dartmouth.edu/

- Johnny Cache for many inspirations
- Joshua Wright and Mike Kershaw for LORCON
- ToorCon & Uninformed.org
- Everyone else who helped
  (including authors of madwifi*, Metasploit,
  Ruby, Lapack and many other great tools)

Contact Information

Institute for Security Technology Studies
Dartmouth College
6211 Sudikoff Laboratory
Hanover, NH 03755

Phone: 603.646.0700
Fax: 603.646.1672

Email: info@ists.dartmouth.edu