Wireless Course: Wi-Fi Tools

These notes are derived from the wireless course run at B-Sides Delaware 2013.

An extra thanks to all involved in the creation and presentation of this course.

Sniffing Wi-Fi

Sniffing Wi-Fi is very simple, if you have the drivers and firmware that are capable of doing it.

Linux is definitely capable of it, with most drivers. BSD is often capable of it, but can be less supported by tools. Windows is only capable with special hardware or commercial software, and OSX can - sometimes.

Mobile devices, as a whole, can not, due to firmware limitations.

Don’t call it promisc mode

Promiscuous mode on wired Ethernet turns off the hardware MAC filter so that it reports all traffic the card sees. It still reports 802.3 Ethernet frames - just more of them.

Monitor mode, or RFMon mode, changes the configuration of a Wi-Fi card so that it is (typically) no longer connected to an access point, and it reports 802.11 formatted packets.

Normal Wi-Fi modes emulate an 802.3 stock Ethernet device, so that the operating system knows what to do with it. A Wi-Fi card in Monitor mode is no longer an ethernet device!

802.11 in normal (or promisc) mode

If you put an 802.11 card in promisc mode (or do a capture on a normal station interface without being in promisc mode) you get 802.3 formatted data frames of the network you’re associated to.

if you’re on an open or WEP network, you will see packets from all users (in range of your card). If you’re on a WPA protected network, you will only see packets destined to your device, and broadcasts.

You *may* see some data frames from overlapping networks, because sometimes, drivers suck.

802.11 in Scanning Mode

Netstumbler, operating system scanning, and mobile tools all use scanning mode.

Firmware in the Wi-Fi card listens for beacons of advertising networks and provides a summary of the visible networks.

There is no way to get packet data or exact beacon records in scanning mode.

This is very noisy.

802.11 in Monitor Mode

In monitor mode, Wi-Fi cards return raw 802.11 frames, such as management frames (beacon, probe, etc), data frames (from all networks overlapping the channel), and control frames like ACK, clear-to-send, and request-to-send.

You may also get a flood of broken crap, like fragments of frames which were correct enough to be detected as an 802.11 frame, but corrupted by noise, another packet causing a collision, etc.

Broken Frames

Normally cards only report valid frames - those which pass the PLCP, the 802.11 packet preamble, and which pass the FCS, frame checksum.

Some devices and drivers will disable this filter when in monitor mode, leading to a flood of crap packets. This can be controlled with the ‘iw’ tool or filtered by validating FCS checksums (Kismet does this automatically)

What Supports Monitor?

Anything using the mac80211-based (ie in-kernel) drivers in Linux *should* support monitor mode automatically.

Anything using out-of-kernel drivers almost definitely will ***not*** work! Vendor-supplied drivers typically do not support monitor mode.

Beware! Many distributions ship out-of-kernel or busted drivers! Anything using a Broadcom chipset is likely to ship with the ‘wl-station’ driver, which will not work. Typically any other vendor driver labeled ‘-station’ will not work.

Atheros is almost universally good, as is the rtl8187 chipset and ralink chips. Broadcom is almost universally bad. When in doubt, buy from a vendor who specifically lists what chipset is in the device they're selling you.

Not just 802.11 ...

There's a lot of meta-data which occurs when capturing Wi-Fi packets, such as channel the radio is tuned to, signal level, antenna status, etc. This isn't part of the 802.11 packet format, so it needs to be stashed somewhere.

This is typically accomplished by the operating system using custom packet headers. There are a number of different packet header formats which can confuse different tools, as not all of them support all the formats.

Radiotap

Radiotap is a "standard" which originated in BSD and has since been added to the Linux kernel mainline.

Radiotap headers include multiple fields which can vary per-packet. It's generally difficult to parse, but contains most of the information possible. Tools are finally consistently supporting it.

When doing monitor mode in Linux, radiotap is what you'll get if you just run tcpdump to capture packets.

PPI

PPI stands for Per Packet Information header, and was originally developed by CACE Technologies for 802.11n capture in Windows via the AirPCAP.

it is well supported in Wireshark, since the core developers work for CACE.

PPI encodes all the information available in radiotap, as well as being able to contain GPS, spectrum samples, and more. Unfortunately, because of some oddities, it is often considered the bastard step-child of header formats, and the tcpdump/pcap developers prefer to not see it expanded to include non-802.11 data.

Pcap-NG

Pcap-NG is a very new format which makes for better pcap files. It allows multiple interfaces to be captured into a single file, file merging, and more. Variable meta-data is encoded as part of the file spec.

Currently, Wireshark handles Pcap-NG, but not much else does. As it is finished, more tools will support it and it's likely to become the de facto new standard for files, but currently it is uncommon.

Configuring Monitor Mode

To start configuring, we'll step through setting up a monitor mode interface. Typically wireless interfaces are named wlanX but newer distributions are changing that.

To list available interfaces:

```
# iwconfig
```

Interfaces displayed as not supporting wireless extensions are, of course, not wireless devices.

Look for interfaces which list IEEE80211 support.

The "iw" tool does (almost) everything when configuring a Wi-Fi interface in Linux. It speaks the netlink protocol for the new mac80211 driver model.

Typically in Linux the first interface of a type detected will be numbered zero, while additional interfaces will be numbered incrementally. If your laptop has a built-in wireless card, it will likely be wlan0. A USB connected interface will likely be wlan1. If not, adjust the following commands appropriately.

```
# iw list
```

Will show detailed information about all Wi-Fi interface.

Wi-Fi Physical Interfaces

Kernel drivers support multiple virtual interfaces per physical interface. Each Wi-Fi card presents a physical interface (such as phy0), a primary virtual interface (such as wlan0) and a number of secondary virtual interfaces.

Make a monitor mode VIF

To make a monitor mode virtual interface:

```
# iw wlan1 interface add wlan1mon type monitor
```

The name ‘wlan1mon’ could be anything, but to make it easy for us to keep track if we have multiple interfaces, we name it sanely.

Did it work?

To see if it worked:

```
# ifconfig wlan1mon

wlan1mon  Link encap:UNSPEC  HWaddr
8C-70-5A-C9-36-44-70-42-00-00-00-00-00-00-00-00
```

Notice the bizarre MAC address? This is because in monitor mode, the type of the interface is no longer Ethernet. Linux doesn’t know what to do with it, but it tells us that it worked. If the interface is not there, something has gone wrong.

Test getting packets

First, capture packets from the non-monitor-mode interface:

```
# tcpdump -eni wlan1
tcpdump: verbose output suppressed, use -v or -vv for full protocol
decode
listening on wlan1, 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
```

Note that tcpdump reports type EN10MB (Ethernet). A non-monitor-mode interface still acts like a normal Ethernet.

Now lets try our monitor mode intf

```
# tcpdump -eni wlan1mon
tcpdump: wlan0mon: That device is not up
```

The new interface hasn’t been set ‘up’. Interfaces start as ‘down’ in Linux. Bringing up a monitor

mode interface is the same as configuring other interfaces in Linux:

```
# ifconfig wlan1mon up
```

Try it again

```
# tcpdump -eni wlan1mon
tcpdump: WARNING: wlan0mon: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol
decode
listening on wlan1mon, 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
```

Notice the link-type has changed to IEEE802_11_RADIO, and we're seeing raw 802.11 packets like beacons. Success!

Channels

Wi-Fi cards can only tune to one channel at a time. However, since channels overlap, you'll see bleed-through from adjacent channels. The amount of bleed-over depends on power, proximity, quality of the transmitter, and so on.

Fortunately, we can change channels run-time.

Looking at multiple channels

Run tcpdump in one terminal:

```
# tcpdump -eni wlan1mon
```

Then, set the channel in another terminal:

```
# iw dev wlan1mon set channel 1
```

What went wrong?

This command might have worked; but more likely, if you've been setting up as we've described so far, it may return an error such as "-14" or "-22".

This happens because Linux tries to be smart; Monitor mode interfaces are "secondary" virtual interfaces, while station (client) or access point interfaces are "primary". When a primary interface is present, Linux prevents a secondary interface from changing the channel, since it would break whatever the primary is doing.

wlan1 is probably a client interface, and is 'up'. To fix this, simply bring the parent interface down:

```
# ifconfig wlan1 down
```

Do it again

Now run tcpdump again in one terminal, and change channel in another. You should see the output vary, depending on your local wireless environment.

Network Manager (can) suck

Network manager and other tools, like knetworkmanager, wicd, dhcp clients, and so on, can cause problems by trying to be smarter than the user. All of these can bring an interface down, or back up again, when you don't expect it.

You can tell networkmanager to ignore your interface, or turn it off entirely while messing with Wi-Fi.

Doing it a simpler way

Many tools will do it automatically for you. Kismet will automatically make a monitor mode VAP when given a normal Wi-Fi interface.

Aircrack-ng ships with a tool, airmon-ng, however it is a bit out of date and doesn't preserve sane interface names. A much better tool, included in Aircrack-ng 1.2 (beta1 and higher) and soon to supplant airmon-ng, is airmon-zc.

To make a monitor mode interface from wlan1 and sniff on channel 11:

```
# airmon-zc start wlan1 11
```

Airmon-zc will detect the driver, warn of errors, and warn of system settings and daemons that might conflict with monitor mode.

Wireshark

Wireshark is more or less the greatest network sniffer there is. Of course it supports Wi-Fi!

Start wireshark and point it at the monitor mode interface you've created. You'll be able to look at all the packet headers, in detail.

If you provide the WEP key or WPA passphrase, Wireshark can even decode the traffic for you.

All the additional dissectors in Wireshark, like following HTTP and other data streams, work with Wi-Fi.

Wireshark Filters

Wireshark has an extremely advanced filtering language, which is excellent for drilling into packet captures.

Browse through the Wireshark filtering options for 802.11; extremely interesting and complex filtering is possible.

Kismet!

Kismet is a wireless-specific sniffer, device discovery tool, and wireless intrusion detection system.

Kismet is getting increasingly simple to configure - it can be done entirely in the UI with modern versions.

If you're not using Pentoo, ***MAKE SURE*** you actually have the latest version by checking the Kismet website.

Getting Kismet running

Kismet can be run as root, but this isn't recommended for security reasons. Like Wireshark, Kismet can work as a non-privileged user by using a small capture daemon. This prevents a bug in the packet processor from allowing root access.

Setting up the user

To allow a user to use Kismet, add them to the ‘kismet’ group:

```
First see if the user is already in the kismet group. The groups can
be checked using the ‘groups’ command:

$ groups
wheel wireshark plugdev wifi

Not yet in the kismet group? No problem, add the user to the kismet
group like this:

# usermod -a -G kismet <username>
```

If the user is currently logged in, they will have to log-out and back into the system. Linux does not update groups until a user logs in. Let’s check groups again:

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

Fire up Kismet!

```
$ kismet
```

… That’s it!

Navigating the Kismet UI

When in the Kismet UI, ‘tab’ switches between buttons and text fields. ` or ~ (back-tick or tilde) activates menus. Arrow keys navigate the menus, and enter selects.

When using xterm or other compatible terminals, the mouse can also be used to select menus, buttons, and text fields.

Auto-configuring interface

Once you tell Kismet the interface (in our case we’ve been using wlan1), it will make the monitor mode interface, and automatically bring down the parent interface so that the channel can be controlled.

Configuring the Kismet UI

Various elements in the Kismet UI can be turned on and off via the ‘View’ menu. Try configuring them to your liking.

Network and device details

To select a network to view details, first Kismet must be set to sort the network list. By default, Kismet tries to display all the active networks, in “Auto-Fit Mode”.

Go to the ‘Sort’ menu and pick something other than auto-fit.

Per-network details

Scroll to a network using the arrow keys and press enter for details. Alternately, double click on a network with the mouse.

Basic network info

The first section of information is the basic network info, such as the SSID, times Kismet has seen it, the channel the network advertises, and so on.

Frequency map

Wi-Fi beacons tell what channel they come from, so that clients can tune to the proper channel. Other packet types do not.

Since Wi-Fi channels overlap, Kismet uses the frequency that the radio was tuned to in order to determine what channel a packet might have been on. The frequency map display shows what channel Kismet was listening to, and can be used to identify interfering networks, weird behavior, etc.

SSIDs

One piece of hardware may advertise multiple SSIDs. Typically a new BSSID (MAC address) is generated for each SSID, but some older hardware uses multiple SSIDs on a single MAC address. If there are multiple SSIDs, the SSID has changed, or a client has used multiple SSIDs, they will be shown.

Signal levels

Signal level reporting is a major point of contention. Many, many drivers report bogus signal levels in monitor mode.

Signal levels may be internally consistent - ie a stronger network will be consistently reported differently than a weak network - but are often not useful between hardware types. Signal levels are almost never useful compared to non-monitor-mode signal levels on the same device.

Clients

Clients can be viewed via the 'View->Clients' menu option, and can be sorted, contain details about each client, and so on. The same types of information are collected for each client as for each network.

Channels view

Kismet offers a holistic view of channel usage on all channels it can see, via the 'Windows->Channel Details'.

Channel details include the average signal level of packets per channel, the number of packets per channel, and the amount of data per channel. In some situations there may be a lot of packets per channel - but not a lot of data in them.

Channels

Remember how we said a Wi-Fi card can only tune to one channel at a time?

Kismet gathers data from multiple channels by hopping very quickly among them. This means that some data is missed, because Kismet is not on every channel all the time, but it offers a fairly complete view of what is going on.

To monitor a single channel, use the channel config option under 'Kismet->Config Channel'. This allows custom channel lists, single channels, and so on to be set.

Uses for Kismet

Kismet is useful for general exploration and wardriving, but also for finding rogue access points that may be on your network, rogue clients connecting to your access points, and detecting wonky business going on.

Kismet as WIDS

Kismet is both a fingerprint (specific bad packet) based IDS, and a trend-based IDS where

normal packets are used to behave oddly.

Kismet can detect spoofed networks via channels changing, timestamps changing, and duped SSIDs coming from unexpected MAC addresses.

Kismet can detect spoofed clients by the same client being present on multiple channels, a client reporting two different operating systems via DHCP, and other characteristics.

Kismet can also detect several denial-of-service attacks such as broadcast disassociation and deauthentication attacks.

Kismet log files

By default on Pentoo, Kismet logs into the /tmp directory.

Kismet generates multiple log files:

The “pcap” log contains standard packet capture data. This can be opened in Wireshark or other tools.

The “nettxt” and “netxml” files contain the network details, in human readable or processable XML format.

Other files like “alert” and “gpsxml” contain additional data.

Kismet and LEO

When running Kismet, there are many legal issues that you should consider, especially that Kismet is a full wiretap and not the same as a pen trap and trace. What this means is that It copies the full packet into memory, which constitutes a full capture.

Even in “don’t be evil” mode (hidedata=true) the data is processed before being discarded, so it captures the full packet in memory, then applies a filter that modifies the packets, and saves the results.

Fundamentally, if you’re logging for evidence, use tcpdump.

Expanding Kismet

Kismet is actually a pair of tools, a client (kismet_client) which presents the user interface, and a server (kismet_server) which does the work. They can be run independently (a dedicated server would only need kismet_server).

It's fairly trivial to write additional clients in various languages. All the language needs to be able to do is talk to a TCP socket.

The Kismet protocol is analogous to IMAP or SQL; it is divided into multiple sentences, which consist of fields. A client subscribes to the sentences and fields it is interested in.

Kismet Plugins

Anything which needs to happen in the Kismet server takes part as a plugin. Kismet comes with a handful of plugins by default, divided into two groups - normal, and restricted.

Normal plugins contain options which generally aren't used by most users, such as support for the Wi-Spy spectrum analyzer. The restricted plugins group generally contain more hostile or questionable plugins, such as WEP tools.

Kismet and WEP

The Kismet Auto-PTW plugin integrates the PTW WEP attack from Aircrack. As Kismet collects WEP packets, it automatically attempts to crack the WEP key every few thousand packets. WEP sucks so much, this eventually succeeds - and the PTW attack is so computationally cheap to perform, it makes sense to just keep trying.

The Kismet Auto-WEP plugin attempts to guess the WEP key based on the SSID. Various router manufacturers have used publicly-discovered algorithms to generate a WEP key from the network name.

Ruby

Many people are big fans of Ruby, and the Metasploit project uses it exclusively. To make it easier for client developers, the 'ruby/' directory of the Kismet source contains a helper library and a number of example clients of various complexity:

kismet_alert_syslog.rb links the Kismet alert system to standard syslog, allowing alerts to be propagated through a standard logging system like other syslog events.

kismet_addsource.rb shows how to programmatically add a new capture interface to a running Kismet server.

kismet_shootout.rb is the most complex example client, which allows the comparison of multiple Wi-Fi cards in monitor mode simultaneously. it will show statistics about how many packets are seen by each, which can show network interfaces that drop packets or lack sensitivity.

Clients vs plugins

Everything show in Kismet currently is done in a client. A custom client can do anything Kismet can do currently, but to expand Kismet beyond that, a full plugin is needed.

Plugins are written in C++, and loaded directly into the client or server code. Internally, Kismet is basically constructed of plugins which are compiled together.

Plugins can define new drivers, handle non-802.11 data, etc.

Kismet future

Kismet is expanding to cleanly handle non-802.11 data such as Bluetooth, custom radio plugins, and so on. Work on this is in the git tree, but is not ready for release yet.

Going on the offensive...

Kismet is completely passive. Moving towards doing active attacks, we'll switch to using the Aircrack-NG suite.

Injecting packets

Most drivers that are capable of monitor mode are capable of some sort of packet injection. Injecting packets involves crafting an 802.11 packet and writing it to a monitor mode interface, which then broadcasts it.

Unfortunately, Wi-Fi cards are predominantly designed to transmit data frames while associated to a network. While connected to a network, data gets an active acknowledgement from the receiver.

When transmitting raw packets, there is no such acknowledgement, and sometimes the Wi-Fi card might not even transmit the packet.

Testing packet injection

Make a monitor mode interface if one isn't there already:

```
# airmon-zc start wlan1 11
```

Find a nearby access point. You can do this using Kismet, or using the simple network display

tool from Aircrack:

```
# airodump-ng wlan1mon
```

Now quit airodump (control-c) and set the channel to match a network:

```
# iw dev wlan1mon set channel 1
```

Or, use the airmon-zc tool to change the channel:

```
# airmon-zc start wlan1 1
```

Now to inject...

```
# aireplay-ng --test -e VICTIM_SSID -a VICTIM_BSSID wlan1mon
```

'--test' tells aireplay-ng to test injection.

'-e' specifies the SSID. This should be the advertised name of the network you're testing against. It is case sensitive!

'-a' specifies the BSSID, or MAC address, of the network you're testing against. It is *not* case sensitive.

'wlan1mon' is, of course, the monitor mode interface we created.

The theory of breaking WEP

WEP is incredibly broken. The more packets seen, the more broken WEP becomes. The more we know about what is in the packets, the quicker we can break WEP.

The old way of breaking WEP required hundreds of thousands of data packets, and could only use a tiny fraction of them. WEP was known to be broken, but practically breaking it was often very difficult.

The new way of attacking WEP is to find a client on a victim network, kick them off the network, and watch when they rejoin.

The first thing a client does when it rejoins a network is to get a DHCP address, and try to find the MAC address of the gateway via ARP. The beauty of ARP is that it's easy to spot (a broadcast packet of a fixed length), it generates a response (new data packets from the network), and many of the bytes are known.

There are many copies

Not only is WEP broken, it assists us in breaking it.

WEP contains no replay protection - there is nothing that indicates a packet has been seen before, which means a single packet can be sent thousands, or tens of thousands, of times. If that packet elicits a response - such as an ARP response from the gateway - it can be used to generate an arbitrary amount of unique data.

Generating enough unique packets to crack a WEP network is now extremely trivial - combined with advanced attacks which can use more of the received packets, we have reduced cracking a WEP network to a matter of minutes.

Let's do it!

This will take a number of terminals.

First, start logging:

```
# airodump-ng --channel 1 --write /tmp/aircrack wlan1mon
```

This sets the channel to 1, and writes the Aircrack data to files in /tmp.

Next, we need to associate our Wi-Fi card to the access point. Some cards have difficulty injecting data packets from a different MAC address, so the easiest way is to link to the network as if we were a valid client. A WEP network allows anyone to join it - WEP only protects the data!

In a new terminal, run:

```
# aireplay-ng --fakeauth 5 -e VICTIM_SSID wlan1mon
```

This performs a fake association every 5 seconds, to a network named VICTIM_SSID (which is case sensitive!), injecting via the wlan1mon interface.

Now we need to find an ARP packet. In yet another terminal, start aireplay-ng looking for ARP packets:

```
# aireplay-ng --arpreplay -e VICTIM_SSID wlan1mon
```

This tells aireplay to look for ARP packets, from the SSID VICTIM_SSID. Like all other instances

of the SSID, this is case sensitive.

At this point, you may naturally get an ARP packet of a client joining the network. If not, you can help things along.

To force an ARP, we need to find a victim station on the target network. Looking at the output of airodump, we need to find a client whose BSSID matches the network we want to attack.

To force a client to reconnect, we basically cause a denial of service. Wi-Fi management frames have no protection, so nothing prevents us from spoofing the access point and telling the client to disconnect. In *yet another* terminal:

```
# aireplay-ng --deauth 15 -a MAC_OF_AP -c MAC_OF_CLIENT wlan1mon
```

This sends 15 sets of 64 deauth packets (just in case 14 sets of 64 isn't enough), spoofing the address of the access point (the BSSID the client is connected to), targeting the client. Make sure to pick a client which is connected to the network, and don't pick yourself!

At this point, there should be a flood of traffic in the terminal running `aireplay-ng --arpreplay`, and the terminal running airodump-ng should show a large number of packets.

Now to finally crack WEP! Fire up a final terminal, and run:

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

If multiple SSIDs are present in the capture, select the target SSID from the list. After a short time, it should have found a solution.

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

Making it easier

There are many tools which are scripted to simplify this process. Now that you know the actual steps involved, explore tools which simplify it, such as 'wifite'

Attacking WPA

There are no significant cryptographic weaknesses in WPA when using the AES algorithm. The TKIP algorithm was designed as a stop-gap measure, and is beginning to show weaknesses.

In general avoid TKIP wherever possible, however if you are not defending a company network or similar, it is likely still sufficient.

Unfortunately, WPA still suffers from multiple problems in its design and implementation by administrators and users.

WPA-PSK

WPA-PSK generates a complex key from a (usually) human-readable passphrase. The key, called the PMK or Primary Master Key, is then used to generate the per-user temporary keys (PTK) each time a client connects to the network.

If an attacker can derive the PMK, and can capture clients connecting to the network, it can derive the per-user PTK and decrypt the data.

The PMK is a combination of the SSID, the length of the SSID in bytes, and the passphrase. This is then run through a repeated SHA1 hash over a thousand times, which is designed to effectively randomize the result, and to be computationally expensive.

In theory, generating the PMK would be so difficult an attacker wouldn't be able to practically approach it.

Unfortunately, or fortunately, rainbow tables happened. Rainbow tables are pre-computed large amounts of hash data that is then storing it in a format which is compact and quickly searchable.

Rainbow tables can be precomputed. For example, taking a million-word dictionary of 8-character passwords and english words, and the top 5000 SSIDs from a site like Wigle, could pre-guess a large number of poorly configured networks.

The quick way

If you've precomputed the network, cracking a WPA-PSK network is nearly instant, limited basically only by the speed of your disk and processor while searching the tables.

Many poorly configured networks are likely to fall victim to these attacks.

The slow way

In theory, it is *possible* to run the hash computations real-time. It's just really, really slow. PMK generation can be accelerated with FPGAs or GPUs, but there are still limits.

The chances of cracking a non-trivial passphrase in the wild are extremely slim - so slim we have never heard of it actually being done.

Being non-trivial

Remember - the PMK is a combination of the PSK and SSID. To pre-compute values, the SSID must be known, and the PSK must be in a list of guessable data.

If your SSID isn't something likely to be known ahead of time, you've slowed down an attacker and forced them to build a custom table.

If your PSK isn't trivially guessable, you've essentially reduced the problem to brute-force, and the chances of an attacker guessing your network at that point are very slim.

To make a PSK non-trivial, thing "pass phrase", *not* "pass word". Replacing characters in a word is trivial for an attacker to do, and most password attacking tools do this already. A much more difficult PSK would be one made of multiple words - a 5 word phrase is simple to remember, but extremely difficult to guess, so long as you *avoid using book quotes*. Recent research has been revealed in cracking pass phrases, but nearly all use book, bible, or movie quotes.

The fatal issue with PSK

There is still one remaining fatal flaw with PSK. In order to duplicate a network, the following must be known:

- The SSID (which is completely public)
- The BSSID (which is completely public)
- The PSK

When a PSK is posted publicly, for example at a conference, or in a coffee shop, or airport, or whatever, all the information an attacker needs to clone the network is available.

Once an attacker can spoof the network, it's trivial to then perform a man-in-the-middle attack on any traffic a user sends over the network.

WPA-EAP

There are multiple alternate WPA authentication methods beyond WPA-PSK. The Extensible

Authentication Protocol, or EAP system, defines several authentication types for Wi-Fi. Of these, the most commonly supported are TTLS, which uses mutual SSL authentication to validate users by certificate, and PEAP, which uses SSL to protect the exchange with a Radius server authenticating a username and password.

Over all these methods are secure, but there are many ways for a client to act incorrectly (or a user to act incorrectly) which can completely compromise them.

What's wrong with SSL?

How many of you have clicked "OK" on a certificate issue for a website? Users click OK, ignoring self-signed cert problems amongst many other ones. If a user can be convinced to accept a spoofed radius cert for WPA-PEAP, then it's game over: The client will happily pass the MSChapv2 credentials and start using the network. This exposes the client to man-in-the-middle attacks by connecting to spoofed networks, and exposes their credentials to offline attacks, which might allow the attacker into other resources on the network.

Even Dumber

Clients can often be even dumber: Many let allow the user to specify no SSL cert at all, and accept any certificate. This is extremely dangerous, and opens many doors for additional risks. Unfortunately, there are some platforms where this is inherent, such as in Android.

Accepting any cert really means there is no security: Without knowing *who* owns the certificate, there is no way to know that the entity getting the passwords is the entity that's supposed to!

Configuring things right is hard, and users will mess up. Properly configuring devices, especially mobile devices and "bring-your-own-device" hardware on company networks, is extremely difficult. The tools are slowly getting better, but right now there are many vectors to attack them.

Directly attacking clients

There are additional attacks that directly target client devices, and clients in general.
One of the most widely demonstrated and popular attacks known as "evil twin" uses a tool called Karma that functions by responded to any probe sent by a client.

Another tool, called Airpwn hijack's TCP streams on open and WEP networks, which is also useful for attacking client devices.

Client attacks are especially effective because while it is possible to defend access points effectively, and clients can be defended on controlled networks, it's nearly impossible for a network administrator to protect a client once it goes on a public, uncontrolled network, such as

at a coffee shop or airport.

Currently the most advanced attacks are against clients, and it is possible to cause a client to bring a vulnerability back to the corporate network. The only absolute defense is to issue all clients cellular data devices and prevent them from using public access networks.

Thanks & Contact information

Thanks to everyone who attended the class and followed up with these notes!

dragorn@kismetwireless.net
@kismetwireless on twitter

rhandorf@handorf.org