Wireless Penetration Testing System

Zhe Wei Chong 40899013

21 Oct 2007

Contents

1 Introduction

Just as technology in electronic communications are developing in an incredibly fast pace, wireless devices are becoming more ubiquitous in our daily lives. In the rush to push smaller, sleeker "gadgets" into the consumer market, much less emphasis is placed in promoting overall awareness in the security of wireless communications. This project aims to inspect the current state of 802.11 wireless security in general by introducing a graphical tool for auditing the security of WiFi networks. PyScanner is a graphical toolkit for scanning for nearby wireless networks, tagging them using geopositioning, recording and analyzing packets sent over the ether, and generating visualizations. We hope that by employing the tool users can gain a good insight in the strength of their own systems and further understanding in wireless security in general.

One of the most major concerns for implementing network security is the protection of data sent over the connection. This is directly related to the three main concepts in information security: *confidentiality, integrity* and *availability*, also known as the CIA triad[1]. Obviously, unlike wired networks WiFi does not enjoy the benefits of medium isolation. There is no simple way to guarantee if a packet received over the ether has been compromised and thus no longer confidential. We cannot conclude if a packet was truly crafted and sent by the intended source, and whether the packet has never been altered. And finally, because it is possible to launch denial of service attacks with 802.11, service availability cannot be guaranteed. In section 2.1 we will briefly discuss each of the attacks, and how they can be detected.

2 Background

2.1 WiFi and Encryption

The 802.11 Wireless LAN (WLAN) standards, also more well known under its nonsensical technology brand "WiFi" or "Wireless Fidelity", was first introduced in 1997 to support medium range transmissions, and then followed by 802.11a in 1999 [2]. 802.11b, also introduced in 1999, soon dominated the market because of much lower prices than 802.11a. Joined by the wide array of consumer devices and computer hardware supporting it soon thereafter, it is now considered the *de facto* standard for WLAN communication. However, its popularity also meant that much of its security weaknesses were quickly exposed, and also prompted

IEEE to begin work on 802.11i to amend its original specifications.

2.1.1 WEP

Security for the 802.11 specification is supported by the Wired Equivalent Privacy (WEP) protocol, which main aim is to provide link-level data being transmitted between wireless stations connected to access points and the access point itself. WEP supports the main concept from the CIA triad, 'confidentiality', with its use of the RC4 symmetric key stream cipher algorithm for cryptographically protecting data. A key stream is generated and then applied to the data when it is transmitted over the network. As for the shared key used, the key size may be 40-bits or 104-bits, which is appended with the 24-bit Initialization Vector (IV) to create the RC4 key. However, the use of RC4 itself becomes the main weakness of WEP.

To check for the data integrity of data transmitted, WEP uses the Cyclic Redundancy Check (CRC) method to see if the message payload has been altered or perhaps damaged during transmission. The CRC value is computed from the payload before transmitting and sent within the encrypted message, only to be compared again on the recipient's side. While simple, CRC itself is not cryptographically secure compared to hashing or message authentication codes implemented in later standards.

Carefully detailing each weakness in WEP warrants an entire paper to itself, so they will only be summarizes as follows:

- The IV field in WEP used for generating the key stream in RC4 is sent in clear text. Its length, being only 24-bit, causes the IVs to possibly be exhausted and reused within 10 hours [3]. The length of the IV is always known to be 24-bits of the RC4 key. By recording just a relatively small number of IVs it is possible to use them to decrypt the cipher text.

- WEP does not provide cryptographic integrity protection. Using the RC4 stream cipher with the CRC non-cryptographic check opens up an exploit which allows attacks to directly modify packets without decrypting yet, and sending it the access point to see if it was acknowledged.

- Clients do not authenticate the access point, thus enabling the possibility of non-legitimate 'rogue' access points and man in the middle attacks.

2.1.2 WPA and WPA2

Wireless Protected Access (WPA) is a standard promoted by the WiFi Alliance intended to mitigate some of the problems introduced by WEP while 802.11i was still being drafted, introducing 802.1x and the Temporal Key Integrity Protocol (TKIP). TKIP generates per-packet encryption keys, increases the IV space to 48-bits, and also introduces a message authentication code, "Michael", to replay CRC. These measures effectively eliminate most of the well known key recovery attacks on WEP, which will be described in the next section, as well as replay attacks depending on CRC.

The WPA2 standard is based on IEEE 802.11i, which in addition to the improvements already subsetted in WPA, also includes a new AES (Advanced Encryption Standard)-based encryption algorithm CCMP to replace TKIP. The main drawback with WPA2 is that AES is computationally expensive, so many older models of wireless devices do not support it.

2.2 Known Attacks

For the scope of this project, we will concentrate more on 802.11b, which is obviously more commonly accessible in public, and cover the offline dictionary attack on WPA.

Threats to 802.11 security can be simply summarized into the four following attack categories.

2.2.1 Sniffing

802.11 security is most vulnerable due to the fact that packets are broadcasted over the air, and thus may be eavesdropped by any person with the means to do so, also known as "sniffing". It is very simple to carry out, yet as a passive attack almost impossible to detect. An attacker may not directly access the contents of an encrypted packet without the decrypting key. Present time encryption schemes are sophisticated enough that it should no longer be feasible to perform brute force attacks and attempt to recover the original key. However, it may be only a matter of time until the attacker has gathered enough information to regenerate the key using known weaknesses in the cipher suites.

By specializing an attack on the RC4 stream cipher to WEP and focusing on statistical analysis on captured ARP (Address Resolution Protocol) packets, The PTW attack described by Tews, Weinmann and Pyshkinhas [4] has proved that it is possible to retrieve a 104-bit key in under a minute.

For WPA, in PSK (Preshared key) mode the encryption mode suffers from the fact that when generating a session key, only the pass phrase used in the calculation is not known by outsiders. While it has been recommended to use a pass phrase longer than 20 characters, surveys have shown that passwords commonly used are usually just one word from a dictionary, or a combination of a few words. Thus, once a WPA handshake has been captured using a sniffing tool, it is possible to perform an offline dictionary attack and attempt to calculate the original pass phrase.

2.2.2 Spoofing

As mentioned earlier, it is only a matter of time until sufficient information has been collected in order to perform the attacks. This process can be further sped up if the attacker actively tries to stimulate traffic in the target network. "Packet spoofing" can be described as an attempt to craft and send a packet with its contents constructed as if from a different person. For example, when performing the PTW attack it is possible to listen for ARP requests sent on the network, and then attempt to re-inject them into the network, also known as 'replaying', so the router will generate an ARP response. ARP packets are easily identified even though encrypted because of their fixed length and certain bytes. By constantly repeating the process the attacker will essentially boost the packet rate on the network, increasing the chance of the related key attack to succeed.

2.2.3 Denial of service

Because 802.11 control packets are not verified for authenticity, it is also possible to perform denial of service attacks on other wireless clients. One way is to spoof and send a 'deauthenticate' packet to a victim, which tells the client it has lost contact from the network, at which point it will try to reconnect. Thus, flooding the client with de-auths will effectively prevent the client from staying connected to the network. Note that for WPA networks this will also trigger the client to re-associate, which sets up the dictionary attack described earlier. Tews et al. also observes that a number of clients will also flush their ARP caches, so the de-auth also helps in triggering new ARP requests [4]. Another non-harmful but annoying attack is to generate fake access points. An example is the FakeAP tool [5], which is capable of generating thousands of non-existant access points by spoofing 802.11 beacon frames. This may confuse wireless clients trying to find the valid access point to connect to.

2.2.4 Man in the middle

And finally, the fake access point attack can be brought further as a more potentially disastrous threat. In popular places it is common for service providers to offer Internet access, which may be completely open to public or requires authentication through a log in page, also known as a “captive portal”. The attacker may attempt to create a mimicking log in page of the captive portal, host it on a computer advertising itself as a fake access point, and optionally actually connect the computer to the Internet. Clients who log on to this access point instead of the correct one will then fall into the whims of the attacker, now set up conveniently as the standard man in the middle.

2.3 Visualization

Several methods exist for tracking and visualizing wireless networks on real world topology. The most common way used by wardrivers is to utilize the Maps service offered by Google [6], where map images can be easily queried using the offered programming interface. By applying fixed points to the preloaded Google maps based on recorded latitude and longitude, one can easily track locations where certain network SSIDs are visible.

However as single points on a flat diagram, simply knowing the last seen location of an SSID would only be useful for relocating the same network later on, but does not present other possibly interest information that would have been gathered while auditing. A user may also wish to know how far does the signal for a given SSID cover and whether signals have been obstructed by topological features in the line of sight, and then adjust the transmission power of a router, reducing its coverage if necessary. This method is best visualized using common heat map images, which aggregates multiple points together according to a strength rating function to generate a diagram that clearly represents signal strength levels for a wireless signal over a piece of land. An example of this visualization was done by the University of Kansas’s Wireless Network Visualization Project [7]. In their research, Netstumbler was used to collect a positions around a building from which the wireless network is located, and then an aggregate signal strength image pattern is generated and imposed onto an aerial photograph.

Other methods for visualizing common networking infrastructure may be applied while auditing wireless networks as well. By analyzing the 802.11 control frames in captured network traffic, it is then possible to observe data being transferred between nodes. With these information, it is trivial to build a dependency diagram or a flowchart using common graphics manipulation tools like GraphViz [8].

3 Project Specifications

3.1 Aims

The main aims of the project are described as follows:

- **Assessing the usability of UMPCs as a wireless auditing platform**
  The Nokia N800 Internet tablet presents a very interesting platform for mobile development in general. As a computing device it has reasonable hardware specifications with both 802.11 and Bluetooth capability. As a development platform its Linux-based Maemo operating system makes it very suitable for porting and integrating with existing software developed for the PC, and also the freedom to tinker and tweak with system internals where required.
  Before the Nokia internet tablet, there were other Windows Mobile and Linux-based handheld devices which were tested for wireless security auditing use, such as the Sharp Zaurus PDA and Compaq IPAQs. Compared to laptop computers, mobile devices and UMPCs generally boast much better mobility and lower power consumption, which may be important factors when testing for prolonged periods 'out in the fields'. Compared against the N800 however, they do not provide as much CPU processing power as the Nokia's ARM processor, and Linux support on the devices have been said to be 'peculiar' [9] - likely more troublesome than it is worth.

- **Develop a GUI application for wireless auditing**
  The popular wireless monitoring tool `Kismet` has been ported for the Maemo platform before, but it uses a classic text-based interface for displaying and controls, which is very unfriendly when using on the tablet's small screen (see figure 1). Therefore, this project proposes a lightweight

graphical front end for `Kismet`, which will allow auditors to more easily gather data to work with.

- **Research methods to visualizing wireless auditing results**
  As described in section 4.2.1, being able to graphically represent captured data can be very helpful while wireless auditing. As part of this project, I will be researching different methods for representing interesting visualizations, and then develop a module that can be used by the GUI application. Data gathered during the development of the project will be aggregated and used for the final prototype of the rendering system.

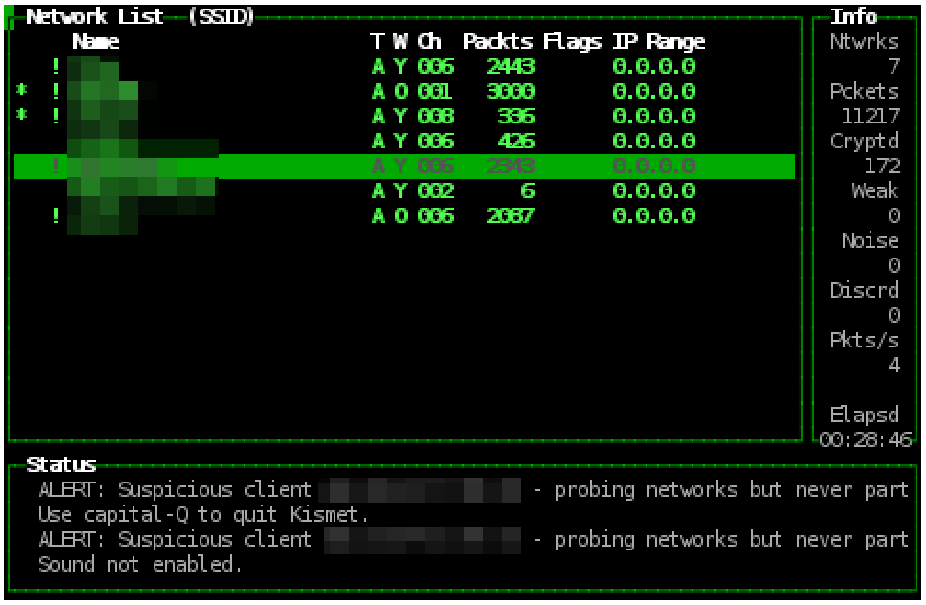
- **Record and observe statistics with regards to WiFi use:**
  Before concluding the project, I will again aggregate all data captured over the lifetime of development and summarize any findings that may be observed from these statistics. These may include wireless usage within the community, observations of WiFi security implementations over the audited area, and the efficiency of wireless penetration testing tools. While generic patterns of these data are usually mentioned in mass media, e.g. that most access points that may be easily accessible in public are still using WEP or no encryption at all, it would be helpful to see some hard statistics, or at least rough estimates of how the condition is in real life.

3.2 Development Methodology

For the development of this project, we will follow the following procedure:

- First, determine important tasks usually done during wireless auditing. Based on the findings, a task-oriented GUI will be developed. When designing the GUI, we will need to keep in mind that the targeted platform has very limited screen size, and will need to design an user interface that is not severely restricted by this limitation.

- As we are using Kismet for its backend, its client server protocol will be analyzed and a prototype that is capable of communicating with it will be developed. We will need to keep in mind that Kismet, as a daemon that requires root privileges to access the wireless card's promiscuous mode, will require different handling than user-space software.



```
Network List  (SSID)                                               Info
     Name                   T W Ch  Packts Flags IP Range          Ntwrks
   ! [redacted]             A Y 006   2443       0.0.0.0                7
*  ! [redacted]             A O 001   3000       0.0.0.0           Pckets
*  ! [redacted]             A Y 008    336       0.0.0.0            11217
     [redacted]             A Y 006    426       0.0.0.0           Cryptd
   ! [redacted]             A Y 006   2343       0.0.0.0              172
     [redacted]             A Y 002      6       0.0.0.0             Weak
   ! [redacted]             A O 006   2087       0.0.0.0                0
                                                                    Noise
                                                                        0
                                                                   Discrd
                                                                        0
                                                                   Pkts/s
                                                                        4

                                                                   Elapsd
                                                                 00:28:46
Status
 ALERT: Suspicious client [redacted] - probing networks but never part
 Use capital-Q to quit Kismet.
 ALERT: Suspicious client [redacted] - probing networks but never part
 Sound not enabled.
```

Figure 1: Kismet's text-based GUI

- Based on captured packets, we will develop a graphical method for visualizing results of wireless auditing on screen. There are many approaches to this problem, therefore we will choose one that can best represent the findings, and making modifications if necessary.
- And finally, we will attempt to analyze all aggregated packets and ESSIDs, and report any observations with regards to wireless auditing.

3.2.1 Software Resources Used

The following table lists all programming related tools used throughout the life cycle of the project.

Category	Software used
Operating Systems	Ubuntu Linux 7.10 and Windows XP
Programming Language	Python 2.5
GUI Designer	Glade 2
Protocol Dissection	Wireshark
Libraries	PyGTK, Scapy
Third party software	Clickmap (David Pardo) [10], PyMapper (Razvan Taranu) [11]
Tools	Scratchbox for Maemo SDK, ImageMagick, Aircrack-NG suite

3.3 Design and Implementation

This section summarizes the implementation details of each component.

3.3.1 PyScanner/Kismet backend

In order to detect and record information from nearby wireless stations, PyScanner uses Kismet as its backend. Kismet itself consists of 3 components: the *kismet_server* daemon that puts the wireless interface into promiscuous mode and processes listened packets using *libpcap*, *kismet_ui* which connects to the daemon and communicates with the server via a predefined client server protocol, and *kismet_drone*, which can be distributed on multiple computers and then polled by the daemon to aggregate the results. For this project, PyScanner will only require *kismet_server* for gathering purposes.

The client server protocol used for communicating with Kismet was dissected and described as follows:

Capabilities: Kismet defines the statistics that can be gathered by each version of the server as "capabilities". For example, each major version may have new capabilities added, or deprecate unused capabilities. When the client first connects, a list of capabilities will be sent to the client as a space delimited string.

Server to client protocol: When started, the Kismet server listens on port 2501 and starts monitoring the wireless interface. When a client connects to this socket via TCP, the server will then start communicating with the client via a predefined line-based protocol. Responses are always sent in clear text using the following format:

```
*{Header}: {Data}
```

The header represents a capability supported by Kismet, and the data is a list of strings delimited by a space. If one of the fields in the data may contain spaces, the field is enclosed with a \001 character. As an example of a response, see the following string.

When the client is connected, first this line is sent to describe the server's version, name and build:

```
*KISMET: {Version} {Start time} {Server name} {Build Revision}
```

The line can be safely ignored, unless we want PyScanner to only use a specific version of Kismet. The next line lists all capabilities of the server:

```
*PROTOCOLS: {List of capabilities}
```

By default, clients will not receive any information from the server other than the TIME response. To actually do so, clients have to explicitly enable the capability by sending the ENABLE command followed by the capability's name and its parameters.

Client to server: Clients always send commands to the server in the following format:

```
!{Command ID} {Capability Name} {Command Arguments}
```

The command ID is an integer that uniquely identifies the command, so that when the server responds or acknowledges the command the client will be able to distinguish it against other previously sent commands. It is similar to sequence numbers in TCP/IP packets. The capability name correlates to one capability currently supported by the Kismet server. A list of commands will be described later in this section. And finally, the command arguments define parameters for the capability. Usually, this is just a list of fields that the client wishes to receive from the client.

Basic capabilities of note supported by the current version of Kismet are as follows:

NETWORK: Represents a single wireless network. This could be an infrastructure access point, or wireless station serving an ad hoc network. Basic fields that will be recorded including network BSSID, SSID, channel number, encryption information, beacon information such as signal strength and noise which is constantly updated, IP address if available, and also total number of link layer control (LLC) packets received as well as data and encrypted packets. Packets received are analyzed on the fly to detect if there are any with weak or duplicate IVs. If an instance of a GPS tracking daemon is available, Kismet will connect to it and link the wireless network with the current latitude and longitude.

CLIENT: Represents a wireless station seen on any of the networks. Clients are represents by the same fields as networks, and may be associated with the wireless network it is connected to when a handshake is observed.

INFO: Represents generic information regarding the current monitoring process. This includes the total number of networks seen, total packets received with a count of encrypted and weak IVs, and signal information.

CARD: Represents a wireless interface currently used by Kismet. This includes interface name as registered by the operating system, current channel, total packets received on this interface, and whether channel hopping is enabled.

ALERT: Represents a suspicious action detected by Kismet's built in IDS system. An example alert is:

REMOVE: Indicates that a handshake has been observed and a probed network should now be shown as an access point. This is usually activated when a client is seen associating with a cloaked access point (one that does not broadcast its SSID), thus uncloaking it.

Having dissected and documented the client server protocol, a parser for the protocol is developed as a module to be used by PyScanner's GUI. The parsing code can be seen in the appendix.

3.4 PyScanner GUI

PyScanner graphical interface is developed using Python bindings to Gtk, the common windowing toolkit used in GNOME. To build the interface, a interface designer called Glade is used to generate a .glade file, which encodes all widgets in XML while preserving hierarchy. The Gtk window is then set up with its children widgets then mapped onto it programmatically with the following code:

```
window = gtk.glade.XML("scanner.glade")
```

A more challenging problem is how to display information from Kismet on the N800's relatively small screen. Considering the major tasks performed when wireless auditing, the main window is divided into different tabs:

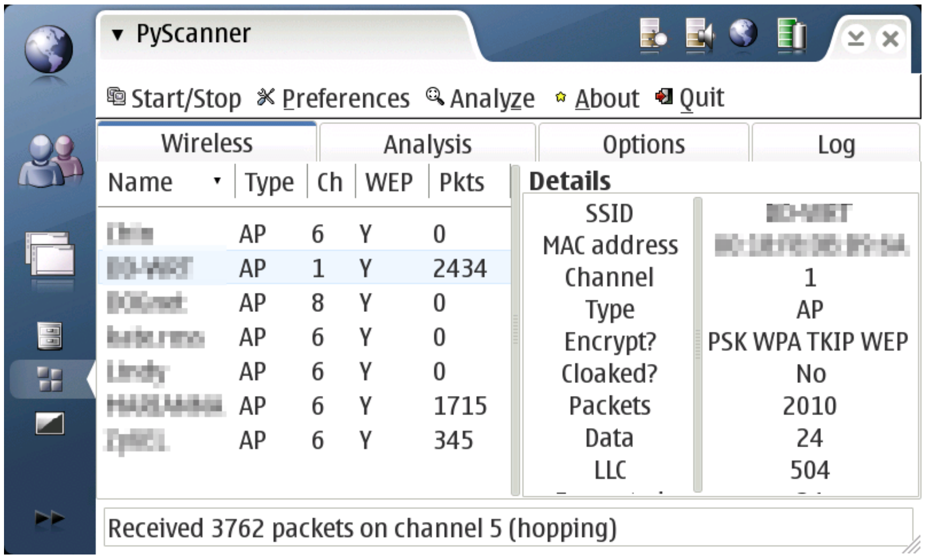
- The "Wireless" tab is the default view of the window.
- The "Analysis" tab is used for post-processing data.
- "Options" is used by reconfiguring PyScanner on the fly, and
- "Log" is used for displaying additional verbose messages.

The Wireless tab itself is split into 2 parts, as seen in figure 2. The left side is a Gtk list view widget displaying a summary of all wireless networks seen, and on the right side more information for an individual network will be shown when it is selected on the list. Because the screen has very limited vertical space, each column on the tree view is set to re-sizable when dragged, and can be sorted in ascending or descending order when clicked. This allows certain common usage patterns such as tracking a particularly ESSID after sorting the list by name, only monitoring the changes on the top of the list after sorting the networks according to last seen, and sorting the list by packets monitored to see which networks are more active than others.

To efficiently present additional information to users, a status bar is added at the bottom of the window for displaying single lines of text, and the "Log" tab is added to present detailed log messages, as seen in figure 3. The status bar actively updates itself whenever wireless information is available and when status changes in PyScanner are detected. Alerts triggered by Kismet is displayed both in the status bar and in the Log tab.

3.5 Visualization

Having gathered enough packets from surrounding wireless networks and tagged them geographically using the monitoring tool, being able to visualize the data is useful for clearly seeing areas that need to be improved and strengthen their security. Originally using a geopositioning system and given the latitude and longitude, one piece of gathered data can only be mapped to a single point on a two dimensional flat map. This does not realistically reflect how wireless signals work. For a wireless access point, the signal is dispersed over a wide



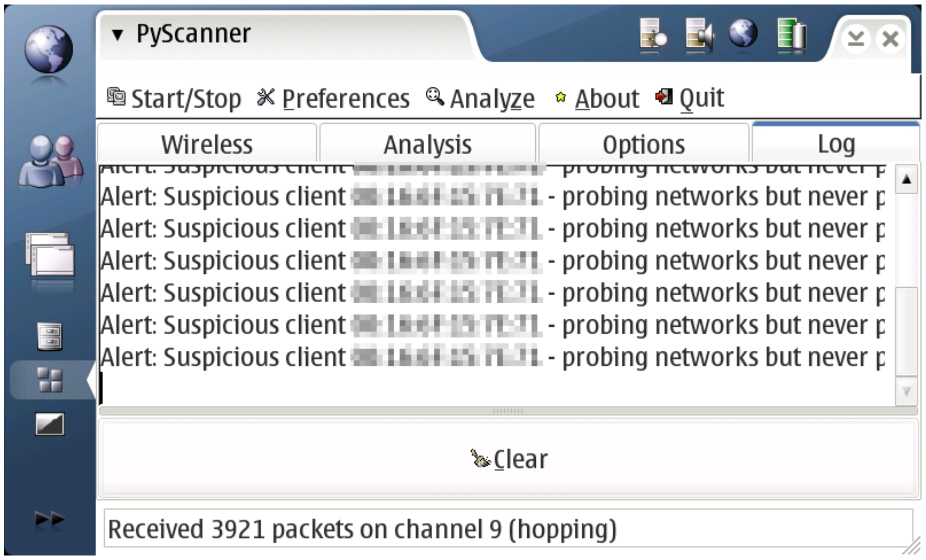
Name	Type	Ch	WEP	Pkts
[redacted]	AP	6	Y	0
[redacted]	AP	1	Y	2434
[redacted]	AP	8	Y	0
[redacted]	AP	6	Y	0
[redacted]	AP	6	Y	0
[redacted]	AP	6	Y	1715
[redacted]	AP	6	Y	345

Details

SSID	[redacted]
MAC address	[redacted]
Channel	1
Type	AP
Encrypt?	PSK WPA TKIP WEP
Cloaked?	No
Packets	2010
Data	24
LLC	504

```
Received 3762 packets on channel 5 (hopping)
```

Figure 2: PyScanner main window, while running



```
Alert: Suspicious client [redacted] - probing networks but never p
Alert: Suspicious client [redacted] - probing networks but never p
Alert: Suspicious client [redacted] - probing networks but never p
Alert: Suspicious client [redacted] - probing networks but never p
Alert: Suspicious client [redacted] - probing networks but never p
Alert: Suspicious client [redacted] - probing networks but never p
Alert: Suspicious client [redacted] - probing networks but never p
```

```
Received 3921 packets on channel 9 (hopping)
```

Figure 3: PyScanner log window, displaying alerts

area depending on antenna direction and signal irradiation patterns, so we can say that a single SSID actually be mapped to multiple points within a region, all together forming an irregular polygon.

If we were to follow that model, this would roughly be the steps followed:

- Every time an SSID beacon is broadcasted its position is logged. Note that for many access points, in order to improve client-side discovery of the access point multiple beacons may be sent per second.
- At the end, aggregate together all coordinates to form a raster structure of the wireless signal map. Alternatively, pick all 'edge' signals and discard all other coordinates in between. This will form a vector structure representing the outer lying edges of the access point signal when its strength drastically reduces.

After experimenting with this model however, it is observed that keeping track of all SSIDs uses up too much space due to a lot of access points broadcasting beacons within very short periods, even up to 200 beacons/second!

Thus, we propose a much easier method:

- Kismet records the minimum and maximum coordinates detected for each SSID. By averaging these two numbers we get a 'best' estimate of the SSID's latitude and longitude.
- Using that as the central point, we simply draw a circle around it. All points are given an equal sized circle, but adjusting its size according to signal strength or other parameters would be trivial.

With these statistics, we can then render it onscreen. A 'heatmap' type visualization was decided, as the color variations drawn over an area is most suitable for showing how the strength of one or more pieces of data can be dispersed over a large area. Eventually, the visualization module was created based on two other GPL-licensed projects:

- 'PyMapper' by Razvan Taranu [11] is a Google Maps browser for Maemo that calculates the current coordinates, downloads the relevant map images from the Google service, and renders it on screen using the PyGame library.

- ‘Heatmap’ by David Pardo [10] is a collection of scripts written in Ruby and Javascript uses for recording the frequency of website visitors clicking on URLs and rendering them as ‘clickmaps’ superimposed over the web page.

3.5.1 Heatmap generation

The steps for generating a network heatmap in PyScanner is summarized as follows:

- First, all Kismet log files that record seen SSIDs are loaded and read to extract a comprehensive list of those SSIDs and their information.
- Next the list is filtered to just keep potentially ‘problematic’ SSIDs. This is defined with the rules below:

  - The SSID does not use encryption, or only uses WEP for encryption.
  - The station is not an access point, but advertises itself uses SSID names such as ‘Free Public WiFi’, ‘Hotspot’ or valid names such as ‘UQconnect’.
  - The station is an ad hoc network.
- The resulting SSIDs are then mapped to a data structure based on latitude and longitude seen. Each mapping is considered a ‘hit’. If another SSIDs have the same latitude and longitude, the ‘hit count’ is incremented.
- Using the aggregated ‘hits’ we can use ImageMagick to render the final heatmap image. First, the number of pixels required for the width and height of the image is calculated. When shown on a map, this also represents the real world distance covered during a wireless auditing session. This is done by finding the distance between an SSID with the lowest latitude and longitude and another with the highest, and then converting it to the units used in Google Maps.
- Now that we have a blank white canvas, each SSID is mapped to a pixel with a gray faded out circle drawn on that spot. Nearby SSIDs will overlap each other forming rounded clusters.

- Next we inverse the colors of the canvas so that clusters of SSIDs are displayed white towards the center and grays out towards its edges. We then substitute each gray shade into a color using a pre-generated color spectrum.
- Finally, the image opacity is adjusted so that it becomes semi transparent, and can be easily superimposed onto other images.

The result of this procedure is our heatmap image, although it provides no geolocation context as to where each hotspot is located. To do so we will draw it in PyScanner.

3.5.2 Google Maps rendering

PyMapper starts by rendering a black screen, and dividing it into tiles of 256 pixels width and height. Google Maps also divides a regional map into cells. According to the zoom level, each cell is then assigned a unique unit number. With these information, PyMapper can then fetch the corresponding map image for each tile and rendering it on screen, requesting more if the user moves the currently viewable screen area and the new tiles were never downloaded before.

To render the heatmap onto PyMapper, the following steps are used:

- First, the current coordinates will need to be determined. As PyMapper does not keep track of coordinates, this is done by checking the center tile's unit number and converting it to a latitude and longitude pair.
- The coordinates that represent the leftmost edge of the heatmap is already known when the pre-rendering calculations are done. Using that, we find the tile number of that latitude and longitude pair, and also the pixel offset within the tile so we can correctly position the heatmap.
- The heatmap is drawn onto the map surface using the PyGame library's API. This will need to be redone every time the map is moved as well.

As a result, we can now clearly see the geolocation context of the heatmap underneath the original image, and can easily move around if the image is very large. An example of the visualization can be seen in figure 4.

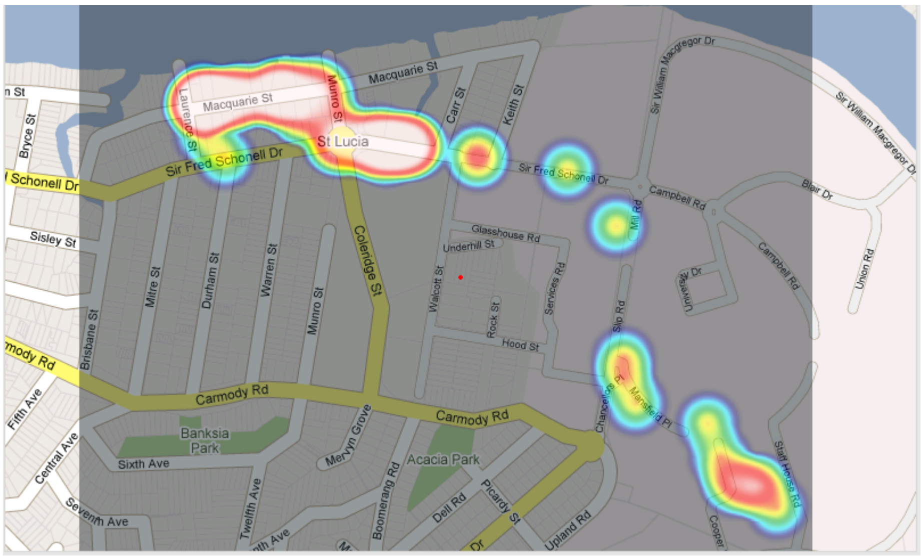


Figure 4: An example heatmap rendered on the path to the University of Queensland

4 Results

This section documents all results, observations and technical insights after the development of the system.

4.1 Observations

After aggregating all monitored data, we find the following observations:

- Out of 1120 access points found, 68% of access points in audited areas are still using WEP. 21% of access points do not have encryption at all, but it may be possible that a portion of them is using alternative security such as MAC address filtering, or are captive portals. 9% uses WPA and TKIP, and 1% use WPA2. Arguably, more users are aware that implementing wireless encryption is important, but do not know which encryption to choose when setting up their network, or simply do not care to change the default recommendations in routers. It is suspected that most off the shelf network routers still do not prominently recommend WPA2 as the default encryption scheme, although we do not have the incentives to verify this.

- A large number of access points, which were not counted, advertised default SSID names such as “linksys” and “Motorola”. This presents a security weakness. We deduce that if the user does not change their access point SSID, there is a high chance that they have not changed the router’s default password as well. Broadcasting the make of the router also enables attackers to narrow down the list of default passwords to test against the router, a resource which can be very easily found on the Internet.
- We also noticed 10 adhoc networks advertising suspicious beacon names such as ‘Free Public WiFi’ and ‘Free Internet Hotspot’. Attempting to connect to one of them did not connect the computer to the Internet, thus we suspect that if it were truly malicious it may have been trying to use other attack vectors, such as exploiting Windows SMB or multicast DNS.
- Attempting to dissect packets for URLs did not yield any interesting patterns, although a number of rather amusing JPEG images were seen being transferred!

To verify the PTW algorithm for retrieving WEP passwords, we also attempted to reproduce the test on a 1Ghz Pentium 3 M computer with a home router. Using the pass phrase “12:34:56:78:90” with a 104-bit hexadecimal key, we attempted to capture as many packets as possible and ran Aircrack with the saved IVs at different intervals.

- Casually browsing Internet web pages produced a very low packet rate, with only 2 IVs being collected after 10 minutes of use.
- Starting an ARP replay attack instantly increases the packet rate to over 500 packets per second. We did not measure the average number of IVs that would have been gathered over that time period.
- With 4000 IVs, 6500 IVs, and then 8000 IVs, the key could not be broken by Aircrack.
- After a much longer interval and at 12000 IVs, the key was recovered in 31 seconds.

Instead of a key with all digits, we also repeated the test with an automatically generated hexadecimal key “B1:DE:A1:79:6C”:

- Again, the key was not broken until at approximately 15000 IVs, where it was recovered in 33 seconds.

4.2 Future Work

PyScanner is planned to be registered as an official community Maemo project some time later. This section details all potential future work on PyScanner, alternative approaches that may be considered, and improvements that should be made that would greatly enhance PyScanner.

4.2.1 Visualization

The algorithm for generating the heatmap with just 100 nodes, even on a 2Ghz Pentium 4 PC, takes 30 seconds to 1 minute. This forced us to separate the rendering code from being integrated into PyScanner, as it will likely take too long on the N800's 400Mhz processor. On the other hand, PyGame is capable of rendering Google Maps rather smoothly at 20 frames per second. It is possible that by moving the heatmap generation process into the PyMapper component it may even be possible to generate live heatmaps.

Also, the information that can be provided by heatmaps is considerably shallow. Even though the code for varying the strength of heatmaps is in place, we did not find a suitable method for ranking different points in the heatmap according to predefined criteria. It may be interesting to define a new ranking algorithm that is capable of assigning a 'hotness' level to points which difference can be clearly seen in the image.

4.2.2 Monitoring

During development a second packet analysis toolkit based on Python and libpcap, Scapy [12], was also evaluated. Scapy proves to be an interesting alternative to Kismet as it allows much more low-level operations during wireless monitoring, for example direct access to packet headers and accessing different layers. Replacing Kismet as a hard dependency will also lighten the load on PyScanner to constantly manage communications with Kismet, as well as removing the need to keep track of Kismet's very sparsely documented client server protocol.

As mentioned in section 4.2.1, using Scapy will allow more interesting ranking criteria, for example, increasing ‘hotness’ whenever:

- Clients are seen accessing log in pages (TCP packets to port 80 which payload may contain the HTTP POST parameters ‘login?username=x&password=y’) without SSL protection, or
- Detecting a large surge of ARP requests and responses from an unidentified wireless station.

Also, it may be useful to develop a Bluetooth scanning feature which listens for Bluetooth advertisement signals from nearby devices. Some GUI elements have been added in PyScanner related to this feature, but in the end due to insufficient time it was not completed.

5 Conclusion

To summarize, wireless auditing provides an insight into how wireless security awareness is still insufficient in the general public. While it appears that most users are know of the ability of using encryption to protect their own privacy and data on wireless networks, many of them are not aware of the dangers and consequences *if* their systems were compromised. From this study in wireless auditing, and proof that an attacker can simply recover the WEP pass phrase within minutes, we can truly conclude that no user with a concern for their own security should even be using WEP any more. And importantly, we would like to emphasise the fact that each packet can be easily ‘sniffed’, and that even with basic security measures in place, there is no excuse to have poor password management practices.

References

[1] C. P. Pfleeger and S. L. Pfleeger, *Security in Computing*. Prentice Hall PTR, 2003.

[2] T. Karygiannis and L. Owens, "Wireless network security - 802.11, Bluetooth and handheld devices," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep., Nov 2002.

[3] A. Bittau, M. Handley, and J. Lackey, "The final nail in WEP's coffin," in *IEEE Symposium on Security and Privacy*, Oakland, 2006.

[4] E. Tews, R.-P. Weinmann, and A. Pyshkin, "Breaking 104 bit WEP in less than 60 seconds," Cryptology ePrint Archive, Report 2007/120, 2007. [Online]. Available: http://eprint.iacr.org/cgi-bin/cite.pl?entry=2007/120

[5] Fake AP. [Online]. Available: http://www.blackalchemy.to/project/fakeap/

[6] R. Gibson and S. Erle, *Google Maps Hacks*. O'Reilly, Jan. 2006.

[7] Wireless network visualization project. [Online]. Available: http://www.ittc.ku.edu/wlan/index.shtml

[8] Graphviz. [Online]. Available: http://www.graphviz.org/

[9] A. A. Vladimirov, K. V. Gavrilenko, and A. A. Mikhailovsky, *Wi-Foo*. Addison Wesley, June 2004.

[10] D. Pardo. The definitive heatmap. [Online]. Available: http://blog.corunet.com/english/the-definitive-heatmap

[11] R. Taranu. PyMapper. [Online]. Available: http://rtaranu.googlepages.com/pymapper

[12] P. Biondi. Scapy. [Online]. Available: http://www.secdev.org/projects/scapy/