Global Information Assurance Certification Paper

Discovering Rogue Wireless Access Points Using Kismet and Disposable Hardware

GAWN Gold Submission

Authored by:
Larry Pesce

Table of Contents

Objective

The objective of this paper is to instruct the reader on how to install and operate Kismet under a Linux operating system, as well as on a Linksys WRT54GL and other disposable hardware. Through the course of this paper, the reader will understand how to use these types of technology to detect rogue access points.

About Kismet[i]

Kismet is an 802.11 layer2 wireless network detector, sniffer, and intrusion detection system. Kismet will work with any wireless card which supports raw monitoring (rfmon) mode, and can sniff 802.11b, 802.11a, and 802.11g traffic.

Kismet identifies networks by passively collecting packets and detecting standard named networks, detecting (and given time, de-cloaking) hidden networks, and inferring the presence of non-beaconing networks via data traffic.

About the Linksys WRT54GL[ii]

The Linksys Wireless-G Broadband Router is really three devices in one box. First, there's the Wireless Access Point, which lets you connect both screaming fast Wireless-G (802.11g at 54Mbps) and Wireless-B (802.11b at 11Mbps) devices to the network. There's also a built-in 4-port full-duplex 10/100 Switch to connect your wired-Ethernet devices together. Connect four PCs directly, or attach more hubs and switches to create as big a network as you need. Finally, the Router function ties it all together and lets your whole network share a high-speed cable or DSL Internet connection. See Figure 1 for an illustration.

Figure 1 – Linksys WRT54GL

About our Other Disposable Hardware

During the course of writing this paper, the author has discovered that the Linksys WRT54GL has not been able to give us all of the proper information in order to accurately locate and recover rogue access points. As a result, we will need to acquire some additional hardware, at low or no cost.

In this case the author was able to acquire a fully working, low end laptop for free from a local company who was going to dispose of it with their local computer recycler. This laptop can be found on E-bay regularly for about $100, with the following specs and illustrated in Figure 2:

Dell Latitude L400

- 700 MHz Pentium III Processor
- 128 Megabytes of RAM
- 8 Gig Hard drive (with appropriate disk wiping)
- External CD-ROM
- Power adapter

Figure 2 – Dell L400 laptop

The author was able to acquire a bundled Dell TrueMobile 802.11b wireless card with external antenna that is suitable for use with Kismet. However, later in this document we will describe an additional appropriate wireless card and additional antennas, at a total cost of about $160.

For purpose of this paper, the author was able to put together a very inexpensive laptop for Kismet, and for connecting to Kismet drones for about $260 including PCMCIA network cards and antennas. With the cost of each drone at about $70, a moderate sized installation of drones and management laptop/recovery laptop are affordable to organizations on a very limited budget.

Installing Kismet on Linux

In this paper we will be installing Kismet on Ubuntu 6.06, the author's Linux distribution of choice. While this installation process may reference this particular Linux installation, the steps should be very similar under all Linux distributions that support Kismet. For installation under BSD derived OSes, OS X, or Windows, refer to the documentation on the Kismet website[iii], as all of these installation methods is beyond the scope of this document.

The easiest way to install Kismet under Ubuntu is via the built in Debian style package manager, *aptitude*. We will illustrate the installation here, but the preferred method will be to install from source, as sometimes the package repositories may contain an older version of Kismet.

To install using *aptitude* under Ubuntu 6.06 we need to perform a few simple steps. First lets make sure that our repository is up to date. With your Ubuntu systems attached to the internet, issue the following command:

```
# sudo aptitude update
```

After the repository has been updated successfully from the internet, we will now need to verify that the repository has updated:

```
# sudo aptitude search kismet
```

This command will output the list of all know package names that contain the phrase kismet:

```
p   kismet              - Wireless 802.11b monitoring tool
```

Now that we can confirm that *aptitude* knows where to obtain kismet, we need to install it:

```
# sudo aptitude install kismet
```

To install from source under Ubuntu, we must first make sure that we have the build environment available so, we will need to perform the following on our internet connected Ubuntu machine:

```
# sudo aptitude install build-essential
```

After we have set up our build environment, we will need to obtain the latest version of the source code. We will use *wget* to obtain the source:

```
# wget http://www.kismetwireless.net/code/kismet-2006-04-
R1.tar.gz
```

Now that we have obtained the source we will need to un-archive the source so that we can begin compiling:

```
# tar –zxvf kismet-2006-04-R1.tar.gz
```

Now we will begin to compile the code to build our executable:

```
# cd kismet-2006-04-R1
# ./configure
# make
# sudo make install
```

Now that we have compiled kismet, we need to perform some basic configuration to make it work in our environment by editing the config file, */usr/local/etc/kismet.conf*, as root. We need to perform the following actions:

- Define a user to drop privileges to by editing the *suiduser* directive in the config file to reflect an appropriate user on this Ubuntu system.
- Set a capture source. We will leave this section alone at the moment, as we will need some additional information from our WRT54GL drone install, which we will illustrate later in this paper.
- Set an absolute path for our output logs by modifying the *logtemplate* directive. The author prefers the logs for kismet to appear in his home directory, so he would use */home/lpesce* in this case. If this directive is not defined, kismet will output the logs in the directory in which kismet was started. This has potential to make a mess of the system by leaving logs all over, which the author can never find when he needs them. An example of this directive would be:

  ```
  logtemplate=/home/lpesce/%n-%d-%i.%l
  ```

Save the config file. We are almost ready to run kismet, however if we were to start it now, we would get an error. We have not defined any capture sources; kismet has no idea at this point where to obtain information about wireless networks. Let's go ahead and build our kismet drone to give it the information it needs.

Installing Open Source Firmware on a Linksys WRT54GL

Fortunately, in 2000, several individuals discovered that Linksys had included open source software in the firmware of the WRT54G line of hardware. As a result, Linksys opened the firmware, and released it back to the community.

The community has taken the open Linksys firmware and developed a number of different forks of the firmware; some good, some not so good. All of the community developed firmware contain improvements in functionality over the stock Linksys

firmware: WPA, increased wireless power, even direct access to the underlying Linux OS.

For the purposes of this paper we will utilize the OpenWRT firmware on the Linksys WRT54GL . In this author's opinion, the OpenWRT firmware is the most robust for our purposes, is easy to use, and will allow us access to the underlying Linux command line. Additionally we will be focusing on the WRT54GL hardware, as this model is easily modified, readily available, and distributed by Linksys for the sole purpose of third party firmware support.

To install OpenWRT on the WRT54GL, we will first need to obtain the appropriate firmware from the OpenWRT website at http://www.openwrt.org. We will be downloading WhiteRussian RC5, with the jffs2 file system. One could utilize the SquashFS file system as well, however the jffs2 file system is easier for us to modify right out of the gate, albeit with some reduced available memory size. The discussion of SquashFS versus jffs2 is beyond the scope of this paper.

Once we have acquired and extracted the OpenWRT firmware, we will need to connect our PC/Laptop to any of the four Ethernet ports (not the WAN port) on the WRT54GL, with the PC set to obtain an IP address via DHCP. Apply power to the WRT54GL, and after a few moments the PC will obtain an IP address. If it does not, we may need to perform a manual IP address renewal. Once an IP address has been obtained, we will note that the PC has been assigned an IP address out of the 192.168.1.0 Class C subnet range, with a default gateway of 192.168.1.1. With this information, we are able to begin the upgrade of the firmware of the WRT54GL.

Open a web browser of choice and point it to 192.168.1.1, where we are presented with the default Linksys login page. We will need to log in to the interface with the default username of admin, and a password of admin. Once logged in to the web interface, we will need to navigate to the Administration tab, and then select Upgrade Firmware. To perform the upgrade, click the Browse button, select the correct OpenWRT firmware from our local disk, click OK, and then the Upgrade button. After a few minutes the WRT54GL will reboot and the upgrade will almost be complete.

One item of note, is that the upgrade will not truly be complete at this stage. With the use of the jffs2 file system, the file system will not be created and ready for use, until a second reboot. After the PC has obtained a new DHCP address after the first reboot, wait one minute, and manually reboot the WRT54GL by removing the power.

After the second reboot, lets begin to use the device, by logging in to the web administration of out new OpenWRT firmware. Direct your browser to the default gateway address of 192.168.1.1, where we will be presented with the new OpenWRT administration website. Now we can set a new password for the administrative user, root. Upon selecting any of the menu options, we will be prompted to provide a new root password. Let us provide a new password of our choice (a strong one of course), and verify. We will now need to re-log into the device as root with our new password.

Before we can continue we need to have internet connectivity to our WRT54G on the Internet or WAN port. When an Ethernet cable is connected, the default configuration of our OpenWRT installation is to obtain IP address information on the Internet/WAN port via DHCP. We will be assuming that you have an appropriately configured DHCP server for this exercise. We do need internet connectivity for our WRT54GL to obtain packages from the internet for installation. There are other methods, but they are not nearly as efficient.

Installing Kismet on the WRT54GL

We now need to begin installing some software on our WRT54GL to make it a bit more useful for our purposes; Kismet, Kismet-drone and wl drivers through "ipkg" packages. The kismet and Kismet-drone packages are what we will use to detect our rogue access points. The wl drivers package includes some wireless utilities that we require for this installation. Unfortunately, the wireless chipset in the WRT54GL does not support channel hopping (scanning though all 12 U.S. channels), so in order for us to perform channel hopping with Kismet and kismet_drone we will need the *wl* driver package in this pursuit, which will be detailed later in this document.

So, we will need to SSH to the WRT54GL as root to install our packages with:

```
# ssh 192.168.1.1 –l root
```

Once connected, we need to perform the following commands:

```
# ipkg update
# ipkg install kismet
```

By installing the kismet base, it will install the other dependencies: *libncurses*, *libgcc*, *kismet-client* and *kismet-server*. We also need to install *kismet-drone*

```
# ipkg install kismet-drone
```

In order to provide channel hopping for our kismet installation, we will need to install the *wl* package:

```
# ipkg install wl
```

We also need to provide a script to automate the channel hopping process, as the broadcom chipset in the WRT54G does not support channel hopping. In order to perform channel hopping, we need to provide a manual method, via a script[iv] in */etc/init.d*:

```
# cd /etc/init.d
# vi S70JW_scan
```

Paste the contents of the following script in to an editable vi window:

```
#!/bin/sh
while : ; do
 wl channel 1 ; sleep 1
 wl channel 6 ; sleep 1
 wl channel 11 ; sleep 1
 wl channel 2 ; sleep 1
 wl channel 7 ; sleep 1
 wl channel 3 ; sleep 1
 wl channel 8 ; sleep 1
 wl channel 4 ; sleep 1
 wl channel 9 ; sleep 1
 wl channel 5 ; sleep 1
 wl channel 10 ; sleep 1
done
```

Save the file and exit vi.

Alternatively we can obtain the script form the internet with wget from Renderman's Renderlab[v]:

```
# wget http://www.renderlab.net/projects/wrt54g/S70JW_scan
```

We also need to change the permissions to the file to be executable at boot regardless of the method in which we obtained the script:

```
# chmod 777 S70JW_scan
```

The script that we have just created will set the channel of the wireless chipset using the wl command every second. We have set it to start up after every reboot, as in this particular type of installation, which is why we have placed it as part of the system startup scripts. If it did not restart after every reboot, we would be only able to monitor one channel, which would not be very useful to us, as a nefarious individual has the ability to set their rogue on any of the 12 available channels at will, so we need to be able to detect on all 12 channels[vi]. We should also note that the author's chipset in the WRT54GL does not support channels 13 or 14 (for use outside the U.S. in limited countries), which is important, as an attacker could potentially violate any FCC (or equivalent regulatory agency) regulations in pursuit of their goals. Unfortunately the WRT54GL would miss APs transmitting on channels 13 and 14 in the U.S., and the author does not condone violating any regulatory requirements.

Configuring kismet-drone on the WRT54GL

Just like the setup under Linux, we need to modify kismet configuration files. Where we will only be having the WRT54GL act in the kismet-drone configuration for AP

detection. In this case we need to edit */etc/kismet/kismet_drone.conf* to define the source for the wireless interface. While SSHed to the WRT54GL, use vi to make the changes to the file by setting the appropriate source:

```
source=wrt54g,eth1,wrt54g
```

Make sure that all other "source" lines are commented out or removed.

We will also need to change the setting to determine which hosts are allowed to connect to the drone via the network. In the illustration here we will set it the drone to allow anyone to connect the Kismet drone on the WRT54GL. Please use appropriate settings for your network, and only allow certain hosts to connect. Hosts can be defined in this setting as comma separated values, with individual hosts, or by network with appropriate subnet mask.

Change the allowed hosts line in the *kismet_drone.conf* to read:

```
allowedhosts=0.0.0.0/0.0.0.0
```

Save the config.

We also need to make some changes to the firewall on our WRT54GL so that we can connect to the kismet drone on the WAN port for our later exercises. We should also allow SSH connections on the wan port as well. We can accomplish these firewall changes by editing */etc/firewall.user*. In this file we need to locate the Open Port to WAN configuration section. In this section we will note that two configuration lines exist for enabling SSH access via the WAN port, but they have been commented out.

Uncomment the two SSH configuration lines in the Open Port to WAN section, and copy and paste a second copy of the two lines. Modify the new second copy of the SSH configuration lines, and modify port 22 to be 3501, the default kismet_drone port. Our final *Open Port to WAN* section should look like this:

```
iptables -t nat -A prerouting_rule -i $WAN -p tcp --dport 22 -j ACCEPT
iptables        -A input_rule      -i $WAN -p tcp --dport 22 -j ACCEPT
iptables -t nat -A prerouting_rule -i $WAN -p tcp --dport 3501 -j ACCEPT
iptables        -A input_rule      -i $WAN -p tcp --dport 3501 -j ACCEPT
```

Save and exit the */etc/firewall.user* file. These changes will be applied after a reboot, but we can make them active immediately by issuing the following command:

```
# /etc/init.d/S45firewall restart
```

We need to make sure that we also start kismet_drone as a service, as we may not have SSH access to the device if deployed remotely. We will need to have access to the drone on TCP 3501, as this is the default port that kismet_drone communicates with, however it is configurable via the */etc/kismet/kismet_drone.conf* file.

To set kismet_drone to start up on boot, use vi to create the file */etc/init.d/S80kismet_drone* on the router with the following:

```
#! /bin/sh
echo "Setting radio for kismet_drone"
mkdir /var/log
/sbin/ifconfig eth1 up
/usr/sbin/wl ap 0
/usr/sbin/wl disassoc
/usr/sbin/wl passive 1
/usr/sbin/wl promisc 1
/usr/sbin/wl monitor 1
echo "Running kismet_drone"
/usr/bin/./kismet_drone -f /etc/kismet_drone.conf >
/dev/null 2>&1 &
sleep 3
echo "kismet_drone now running"
```

We also need to make sure that the file is executable:

```
# chmod 777 S80kismet_drone
```

The script will do a number of things for us to set up the WRT54GL to allow kismet_drone to run, and to protect the user from any legal entanglements. The first two steps make sure our environment is sane, by making sure our log directory exists, and sets the wireless interface into the up state. Then the script performs a number of actions to the wireless:

- *wl ap 0* sets the WRT54GL in client mode to prevent unauthorized access to the client wired network
- *wl disassoc* clears any incidental network associations that could affect scanning results.
- *wl passive 1* puts the built in scan engine into passive mode to prevent from connecting to any wireless networks as a client. This prevents accidental traffic capture on wireless networks that we are not authorized to connect to.
- *wl promisc 1* places the wireless interface into promiscuous mode just for good measure, as we don't want to miss any appropriate wireless traffic.
- Finally, the script starts kismet_drone using the configuration file that we have modified for our installation.

Now that we have completed the setup of our drone, we should test that our configuration survives a reboot. Reboot the WRT54GL by either removing the power, or by issuing the reboot command via SSH. Once the reboot has completed, SSH to the WRT54GL as root, and let us verify that we have our scripts running:

```
# ps –ef
```

This command should return the following output:

```
  PID  Ui    VmSize Stat Command
    1 root       368 S    init
    2 root           SW   [keventd]
    3 root           RWN  [ksoftirqd_CPU0]
    4 root           SW   [kswapd]
    5 root           SW   [bdflush]
    6 root           SW   [kupdated]
    8 root           SW   [mtdblockd]
   49 root           SWN  [jffs2_gcd_mtd2]
   67 root       348 S    logger -s -p 6 -t
   69 root       368 S    init
   70 roo        344 S    syslogd -C 16
   72 root       304 S    klogd
  330 root       328 S    wifi up
  425 root       380 S    udhcpc -i vlan1 -b -p /var/run/vlan1.pid -R
  441 nobody     424 S    dnsmasq -K -F
  447 root       400 S    /usr/sbin/dropbear
  448 root       380 S    httpd -p 80 -h /www -r OpenWrt
  457 root       260 S    telnetd -l /bin/login
  458 root       344 S    crond -c /etc/crontabs
  846 root       600 R    /usr/sbin/dropbear
  859 root       520 S    -ash
 1454 root       748 S    /usr/bin/./kismet_drone –f /etc/kismet/kismet_drone
 1462 root       388 S    /bin/sh /etc/init.d/S70JW_scan start
12254 root       244 S    sleep 1
12255 root       360 R    ps -ef
```

We will note that in this case, our two *init.d* scripts have restarted after a reboot and are running under PIDs 1454 and 1462. Now that we have successfully created our drone, let's configure kismet under Ubuntu to use it.

Configuring Kismet Under Ubuntu with a Kismet Drone

Earlier in this paper we discussed defining a capture source in our kismet configuration under Ubuntu. You will recall that we left the source alone in order to obtain more information about our drone, which we have now built on our WRT54GL.

On our Ubuntu installation, we now need to edit */usr/local/etc/kismet.conf* to define an capture source of our WRT54GL. By defining this source, we will be telling kismet to obtain information about the wireless networks from the WRT54GL drone via the network, as opposed to a wireless card located locally to our Ubuntu installation. The wonderful thing about this set up is that we have the ability to monitor wireless networks from a remote location via a wired network. Sure, we can accomplish this over our own LAN, but this could also be accomplished via the internet as long as we have the IP of the WRT54GL, and can open a single TCP port to the drone.

So, in our installation we need to define a *source* directive in */usr/local/etc/kismet.conf*:

```
source=kismet_drone,192.168.1.1:3501,drone1
```

This defines our source as a kismet_drone (so that kismet can appropriately decipher the data), the IP address and port of our drone (in this case 192.168.1.1 on port 3501) and an arbitrary name.

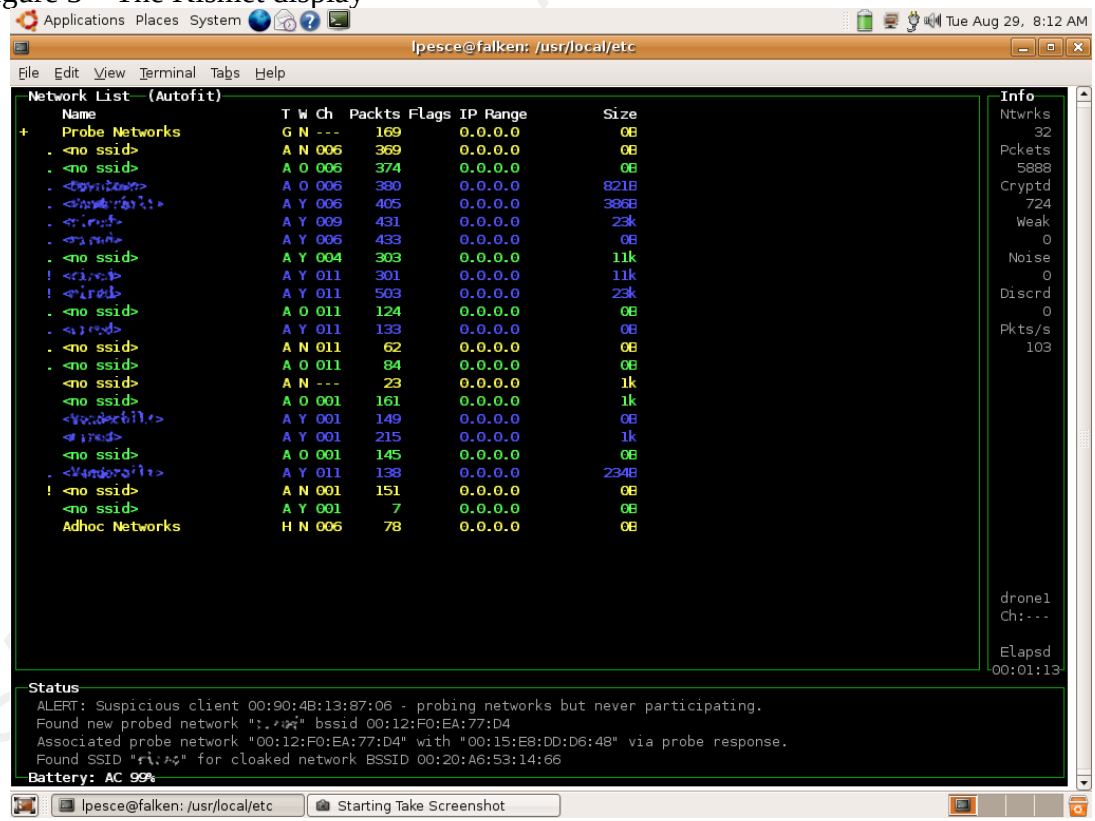
It is possible to have more than one input source. You could obtain information from multiple drones as well as a local wireless card simultaneously. This would be extremely helpful if one were to install an entire network of drones in a facility. We will cover this a bit more in depth later in this chapter.

Now that we have an appropriate capture source defined for Kismet, we need to verify that our setup is functional. Make sure that the WRT54GL is plugged in, and your Ubuntu computer is attached to a LAN port on the WRT54GL. Once you have obtained an IP address via DHCP from the WRT54GL, lets start kismet:

```
# sudo kismet
```

Kismet should now connect to the drone running on the WRT54GL and display the information that it is obtaining about wireless networks on the Ubuntu computer as indicated in Figure 3.

Figure 3 – The Kismet display



Obviously your discovered wireless networks will be different, and the network list has been sanitized. Now that we have Kismet working with a drone, we will also need to make it work with our Ubuntu installation so that we can accurately recover rogue access points.

Using Kismet Under Ubuntu

Previously in this paper, we have installed Kismet under Ubuntu and set up a capture source to use a kismet drone. However, we will also need to set up our Kismet installation to read from a local source.

In this example we will be using a wireless card in our laptop with an Atheros based chipset. In this case we will be using the 300mW a/b/g from Ubiquiti Networks[vii]. I have previously configured this card under this Ubuntu[viii], and a discussion of setting up various wireless cards is beyond the scope of this paper.

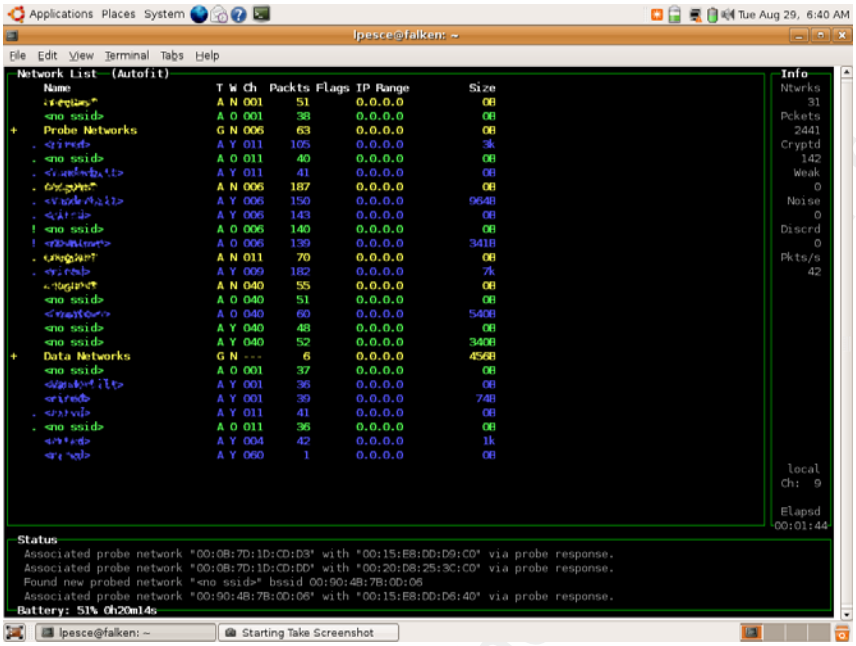
For this installation, we will need to add an additional capture source to the */usr/local/etc/kismet.conf* file:

```
source=madwifi_ag,ath0,local
```

This source defines the driver to use, the BSD style interface, and a unique name. We've elected to use the *madwifi_ag* driver (as previously configured) which will utilize the A, G and B bands, contrary to what the name implies; the G portion of the driver actually utilizes both B and G bands simultaneously. Additionally, the ath0 interface is the appropriate interface for this card for use with our drivers.

Before we start Kismet, we can make some modifications to the default GUI layout to make recovering rogue APs easier. We will note the default display in Figure 4.

Figure 4 – Kismet default display



You will note that in the default display that there is no indication of wireless signal level. We are able to obtain the wireless signal level for each AP that is discovered with Kismet, by performing the following in the Kismet GUI:

- Display the *sort discovered networks list* menu by pressing *s*, and then sorting the networks as you see fit. The author prefers to sort by SSID, by then pressing *s*. Press *q* to return to the main display
- Use the arrow keys to highlight a network in question, and press the *l* key to display signal information. After reading the information and noting the graph, press *q* to return to the main display.
- Use arrow keys to highlight the next network in question…
- Lather, rinse and repeat as needed.

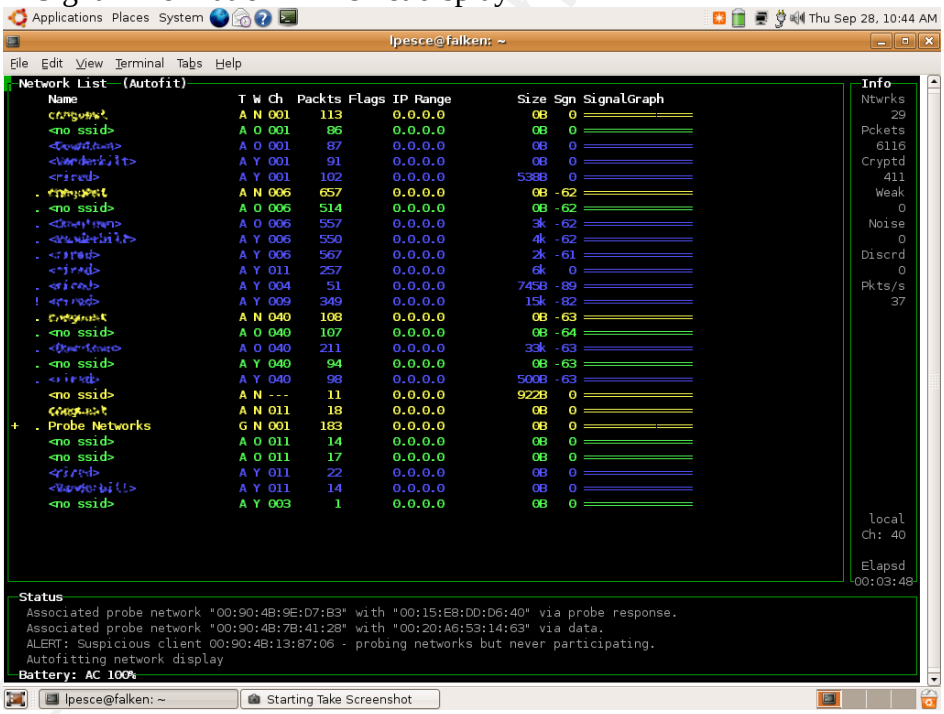
As you can see, the process for obtaining signal information from multiple access points can be tedious. We can modify the default to display our most needed information by adding the SNR (Signal to Noise Ratio) figure, and the graph to the default display. As we will be using the wireless signal level (via SNR) and the signal graph to recover rogue APs, it will be much easier to have that information displayed up front.

In order to make our display more helpful to us we will need to modify the default layout of the Kismet display. We can accomplish this by editing the */usr/local/etc/kismet_ui.conf*. In this file, we will make note of the column configuration directive as this line defines the columns that are displayed. To the end of the *column* directive line, add *signal* and *signalbar* separated by commas so that it looks like this:

```
columns=decay,name,type,wep,channel,packets,flags,ip,size,
signal,signalbar
```

Now that we have modified how our display will look, restart Kismet, and we will note that the display has changed in Figure 5.

Figure 5 – Signal information in Kismet display



Now let us put all of this information together and get hunting some rogue access points!

Issues with Detecting Rogues with the Linksys WRT54GL

Unfortunately, the author discovered during the research for this paper, that we cannot use the WRT54GL for actually recovering rogue APs. In order to accurately locate the rogue APs we need to have signal strength information that we just configured in the last section under our Ubuntu installation. As the author discovered, Kismet on the

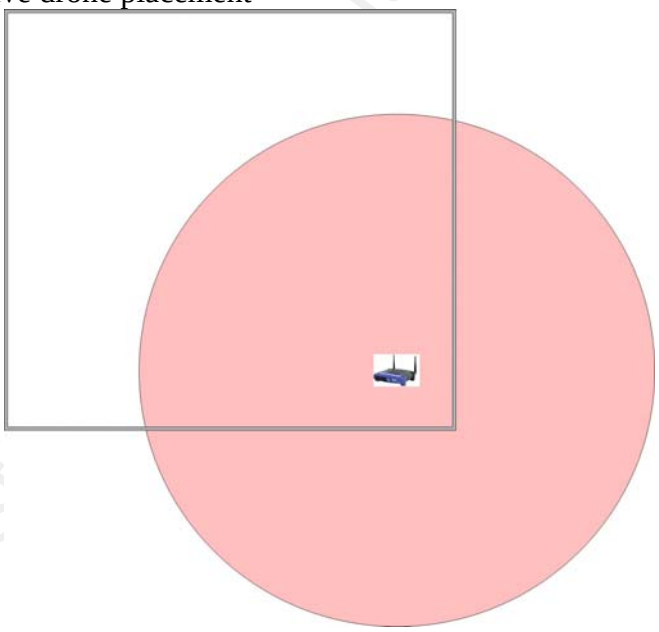
WRT54GL is unable to gather signal strength information! This limitation is due to the nature of the drivers for the Broadcom wireless chipset on the WRT54GL. Broadcom has not released the source code or enough design specifications to determine signal strength from the chipset. The binary driver available from either Linksys or Broadcom does not support signal strength information, or the community is unable to determine signal strength.

However the WRT54GL is still valuable in this exercise. Due to the low cost, small form factor and ease of use, we can leverage one or more of these devices set up as drones to detect a new AP entering our environment.

Drone Placement for Maximum Coverage

When we place a drone, we will want to place the drone in an appropriate place to maximize our coverage. Every place that is not covered by the wireless signal of our drone is a place in which a drone could possibly be placed and would be undetectable. Let us take this simple example under consideration as indicated in Figure 6.

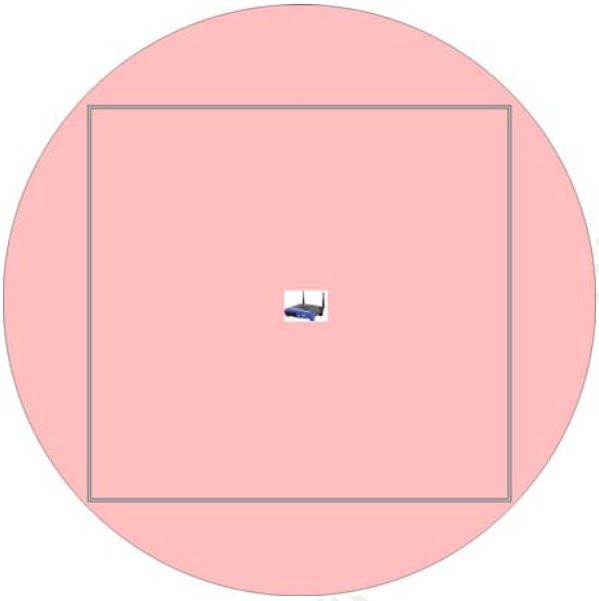
Figure 6 – Ineffective drone placement



In this example, we have an empty square room with our drone placed in the lower right hand corner. While this may be a convenient location for placement, it is clear that we do not have coverage (as indicated by the red circle) in the upper left corner of our room. In the area that is not covered, it would be possible to install a rogue access point, and have it go undetected by our drone. It is also good to note that the drone has a rather large wireless footprint outside of our structure, which may have value in some applications.

If we were to centrally locate our drone, we could more effectively cover our room as illustrated in Figure 7.

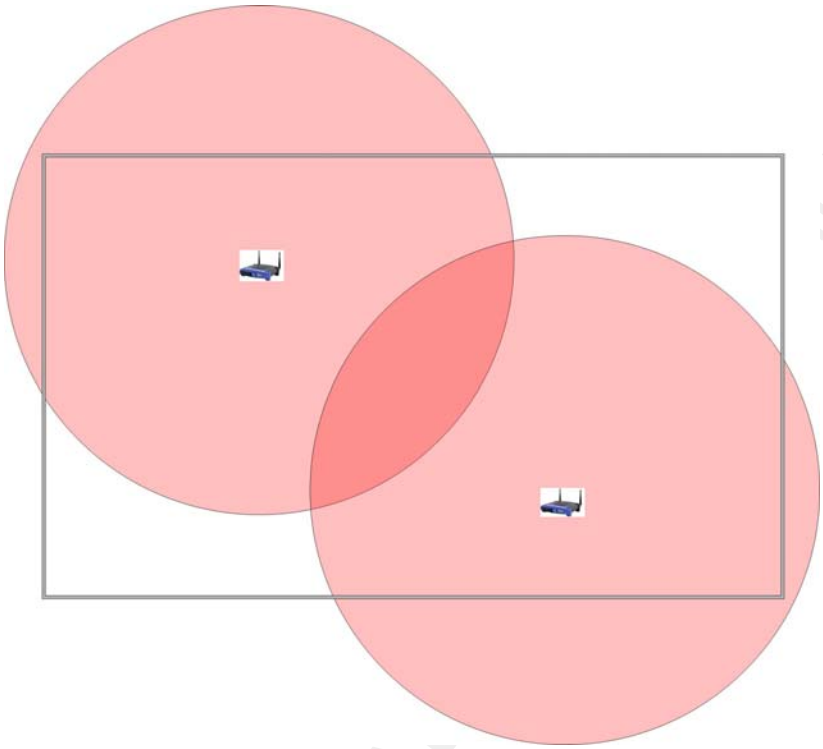
Figure 7 – Effective placement of a drone



After moving our drone, we can clearly see that we have achieved full coverage inside our room.

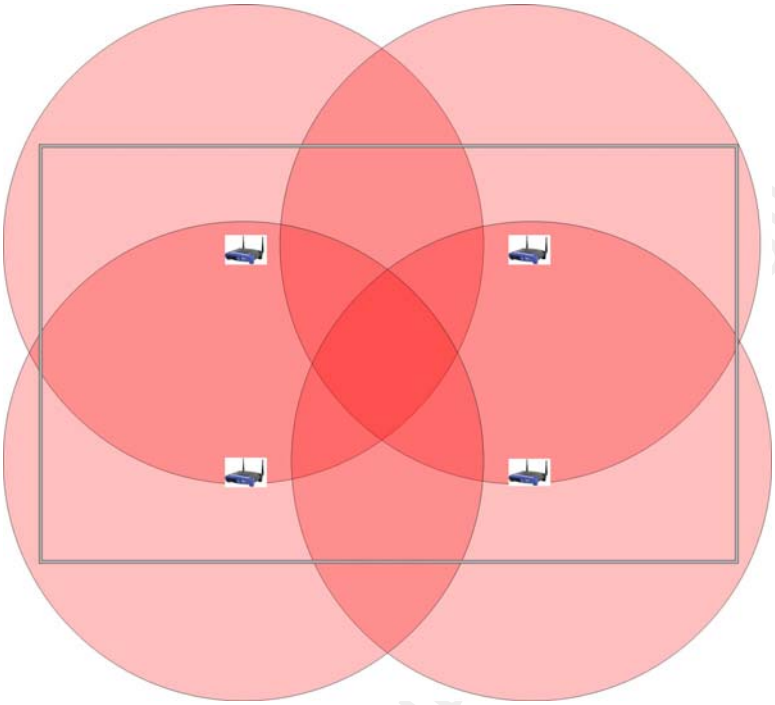
This example is cut and dry. However there may be instances in which we are not able to cover an entire space with one drone. Figure 8 indicates that we have a much larger, empty room with multiple drones placed in opposing corners.

Figure 8 – Drone placement in corners of larger space



Again, in this illustration of our larger room, we have some gaps in coverage with our drones. A more effective solution for this room might be to increase the number of drones to four as indicated in Figure 9.

Figure 9 – Coverage with 4 drones



In this larger room with the addition of more drones as indicated in Figure 7, we have completely covered our room with signal with plenty of overlap, and some significant bleed over into the exterior of our structure.

It is important to note in these examples we are illustrating absolute ideal conditions. The signal of our drones in these environments has not been affected by placement of furniture, elevators, and walls that may reduce coverage. Additionally, the exterior of the structure would impose some signal loss to the bleed over ranges. When faced with a real world installation, it would be advisable to perform extensive testing in the real world environment to achieve appropriate coverage.

With the installation of four drones, we have increased our costs, however we have also reduced our risk. In all environments, it may be possible to have some risk involved at a reduced cost. The balance of risk versus cost should be analyzed for each real world installation.

Installing WRT54GL Drones in a Production Environment

One needs to exercise caution when installing our WRT54GL kismet drones in a production environment. These devices are access points after all, and they should be treated as such. While the wireless interface on these access points are active, you will recall that they are constantly channel hopping. While this will certainly make it difficult (if not impossible) for an attacker to connect to our drones via wireless, it is possible for a failure to occur, thus making this access point another method of access to the wired network.

During our installation of drones, we should also take into account the two Ethernet segments that exist on our router; the LAN and the WAN. Both of these segments can cause different installation issues. Regardless of the method of installation the Ethernet side of our drones should be attached to a network separate from the production network, whether via a standalone air-gapped network, or on a DMZ with limited or no access to other networks. Also, do not forget to account for an additional Ethernet port for our Ubuntu laptop that will act as our monitoring station.

When using the Ethernet port for connection to our standalone drone network, we need to note the default IP configuration of the WRT54GL. If we are attaching multiple drones, note that the default configuration of the drones is to have a LAN IP address of 192.168.1.1, and are set to deliver DHCP addresses on the LAN interfaces. This can cause issues with duplicate IP addresses, and issues with multiple DHCP servers handing out the same address pool on the same network segment.

If you wish to implement multiple WRT54GLs on the drone network, you will want to change the LAN IP and disable the DHCP server on one or more devices. In order to change these settings, enter the commands below as needed.

To change the IP address of the LAN interface of the WRT54GL, perform the following via SSH and inserting the appropriate IP address and subnet mask where appropriate:

```
# lan_ifname=br0
# lan_ifnames="vlan0 eth2"
# lan_proto=static
# lan_ipaddr=x.x.x.x
# lan_netmask=x.x.x.x
```

To disable the DHCP server, edit the */etc/dnsmasq.conf* and comment out the dhcp-authoritative line in the *enable dhcp* section. The new *enable dhcp* section should look like this:

```
# enable dhcp (start,end,netmask,leasetime)
# dhcp-authoritative
# dhcp-range=192.168.1.100,192.168.1.250,255.255.255.0,12h
dhcp-leasefile=/tmp/dhcp.leases
```

LAN DHCP will be disabled after a reboot, or we can force it to disable by restarting the service:

```
# /etc/init.d/S50dnsmasq restart
```

In the other scenario, we may wish to connect the WRT54GL drones to the protected drone network via the WAN port. This configuration will provide more security from attacks via the wired network, should anyone gain access. As you may remember, during the configuration of our drone, we modified the *iptables* firewall rules for the WAN port, and these *iptables* rule sets can be as strict as you desire.

Keep in mind, that when connecting via the WAN port, that by default it expects to obtain IP information via DHCP. If there is no DHCP server on the protected drone network you will want to configure the WAN IP addresses for a static IP address. This can be accomplished by issuing the following commands to the WRT54GL via SSH, and inserting the appropriate IP address and subnet mask where appropriate:

```
# wan_ifname=vlan1
# wan_ipaddr=x.x.x.x
# wan_netmask=x.x.x.x
# wan_proto=static
```

Detecting Rogue APs with the Linksys WRT54GL

As indicated earlier in this paper, there are some significant problems with recovering rogue access points using the Linksys WRT54GL as of this writing. However it is possible to detect rogue APs with the WRT54GL as a kismet drone.

We will need to set up our kismet drone(s) in our environment with appropriate coverage, and connect our Ubuntu laptop to our protected drone network. We can use our existing configuration on our Ubuntu laptop as both a monitoring station and as a rogue APs recovery unit. We will need to make a few additions to our *kismet.conf* to ensure that we have defined all of our capture sources.

We will verify that we have 4 drones defined as capture sources, and one local capture source in our */usr/local/etc/kismet.conf*. Our capture sources should look something similar to this, with IP addresses as appropriate for our installation:

```
source=kismet_drone,192.168.1.1:3501,drone1
source=kismet_drone,192.168.1.2:3501,drone2
source=kismet_drone,192.168.1.3:3501,drone3
source=kismet_drone,192.168.1.4:3501,drone4
```

Now that we have all of our capture sources defined in our master *kismet.conf* under Ubuntu, we need to create configurations for use when monitoring drones and for recovering rogue APs with our laptop. Let's make two copies of our *kismet.conf*:

```
# sudo cp /usr/local/etc/kismet.conf
/usr/local/etc/kismet_monitor.conf
# sudo cp /usr/local/etc/kismet.conf
/usr/local/etc/kismet_recover.conf
```

Now we will need to use both configuration files for our dual purpose. In *kismet_monitor.conf*, we will remove the local capture source, and leave the drones. We will do exactly the opposite in *kismet_recover.conf*, as we will leave the local source, and remove the drones.

We now also will want to be able to start kismet in our two different modes. In order for us to start kismet, we will need to specify command line options to use the different configuration files. The following line will start kismet to monitor our drone installations

```
# sudo kismet –f /usr/local/etc/kismet_monitor.conf
```

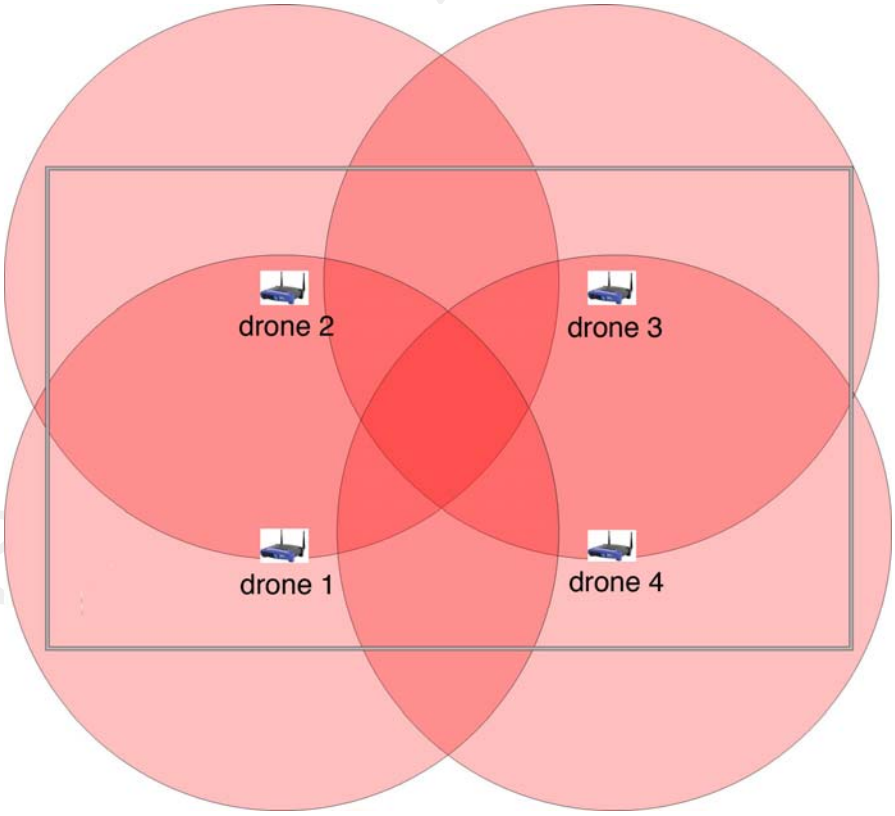
We can start our recovery efforts with kismet by starting it with:

```
# sudo kismet –f /usr/local/etc/kismet_recover.conf
```

In order to determine which drone has spotted a rogue AP, we need to perform some command line and or scripting magic. Unfortunately, when using multiple drones attached to a single monitor, we are unable to determine which drone has actually seen an access point. All data compiled at the monitor location has no concept of which drone it originated at. Conversations with the Kismet developer, and on the forums seem to indicate that gathering this information is unlikely to happen.

However, in a multiple drone scenario, we are able to have some workarounds. First, let us assume that we have four drones located and labeled as indicated In Figure 10.

Figure 10 – Four labeled drones



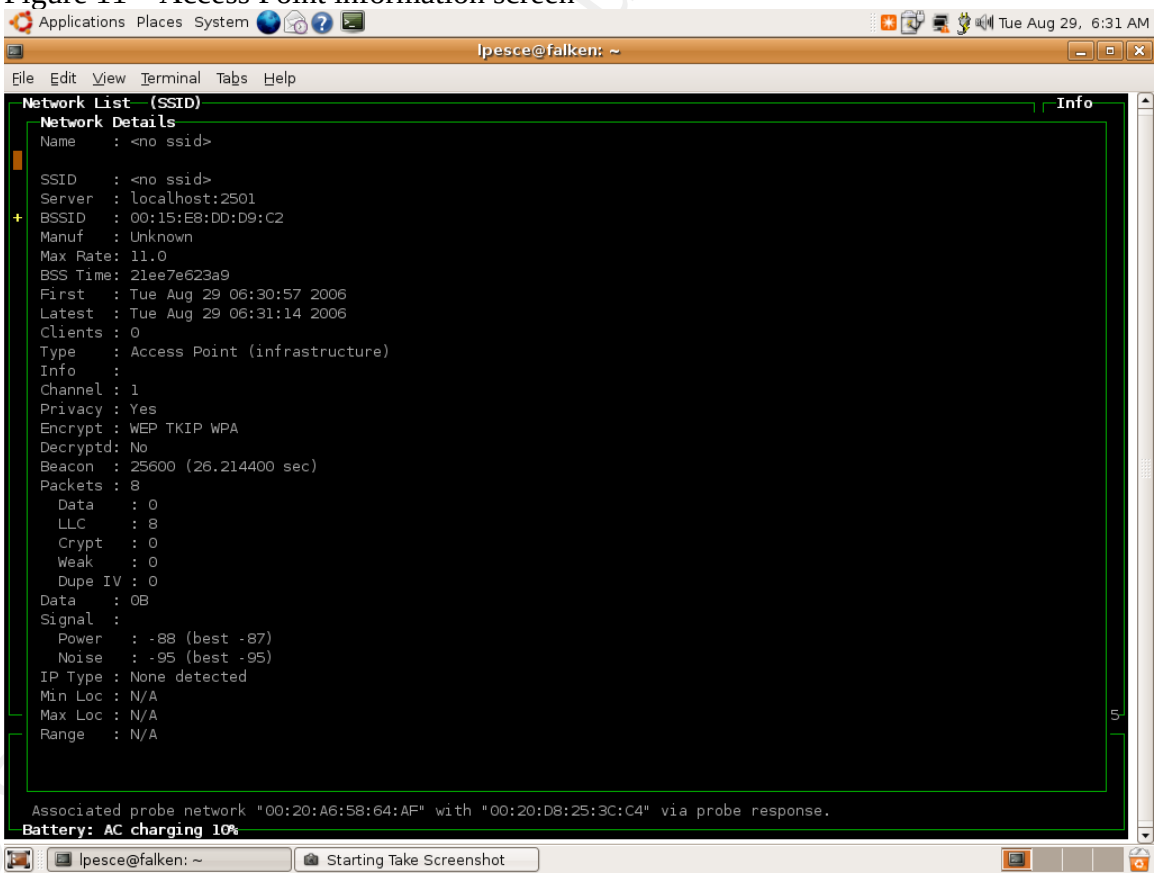
In this scenario, we will need to create several *kismet.conf* files in */usr/local/etc*. We will create each *kismet.conf* with a single source entry; one for each drone in our environment.

As needed, we will exit our kismet monitor when a rogue AP is discovered, and cycle through restarting kismet for each drone and searching for our rogue AP.

There are a few problems associated with exiting our multiple drone monitor set up and cycling through our drones to re-locate a rogue AP. If a period of time has passed from when we discovered the rogue AP to when we take action, the rogue AP may have a cloaked SSID (the SSID is not broadcast), which Kismet would have initially displayed as <no ssid>. However, kismet acts in a passive manner, so the longer it listens for wireless traffic in a given location the more information that it can gather. So, if a client were to associate to the rogue AP, kismet would then be able to determine the SSID as that information is transmitted over the air in the clear, which kismet can interpret.

When we exit the kismet monitor connected to all of our drones and cycle through our drones one at a time, we have effectively cleared our display, and have lost any information gathered from cloaked rogue APs. It is important to note that before exiting the multiple drone monitor to record the MAC address of the rogue AP. This can be obtained by scrolling through the list with the arrow keys to the alleged rogue AP, and entering *i* on the keyboard to display the AP information screen as shown in Figure 11.

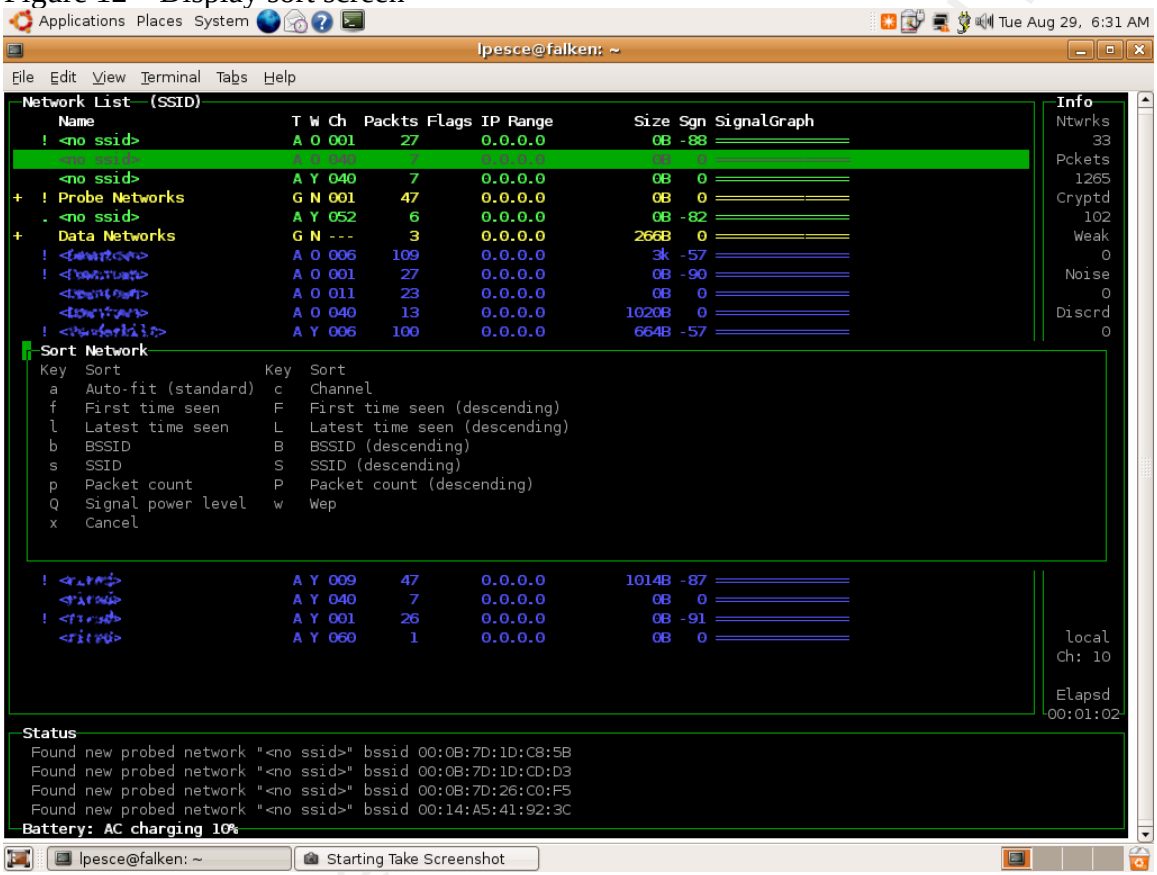
Figure 11 – Access Point information screen



It may be required to sort the wireless network list before it is possible to scroll through the wireless networks with the arrow keys. This is accomplished by performing a sort on

the wireless network display, effectively disabling the *autosort* feature (which is the default sort method). Sorting can be performed by typing *s* at the main display which will display the sort method selection as shown in Figure 12.

Figure 12 – Display sort screen

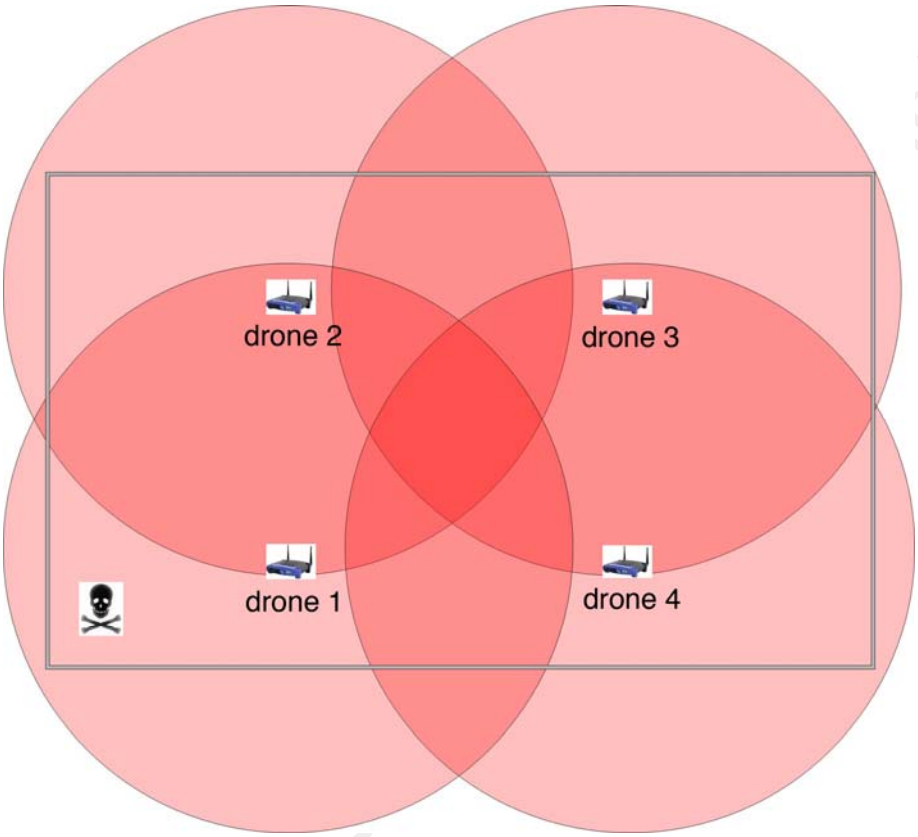


The author prefers to sort the network list by SSID by typing *s* in the sort method display, but you should choose a sort method that is appropriate for you.

Now that we have our rogue AP MAC address recorded, we will begin cycling through our drones. Upon connecting to each drone, we will be presented with a list of wireless networks. If the rogue is not displayed by SSID (even after a sort), we can probably assume that the rogue has been cloaked. This is the exact reason that we have recorded the MAC address of our rogue, as we are able to obtain additional information about all of the APs discovered by kismet, even for APs that are cloaked. We can sort our wireless network list in Kismet, and examine the additional information (as described earlier) for each cloaked AP. With this additional information, we are able to determine if they are a match to the previously discovered rogue by comparing MAC addresses.

Let us assume that the drone has been placed in one corner of our building as indicated in Figure 13.

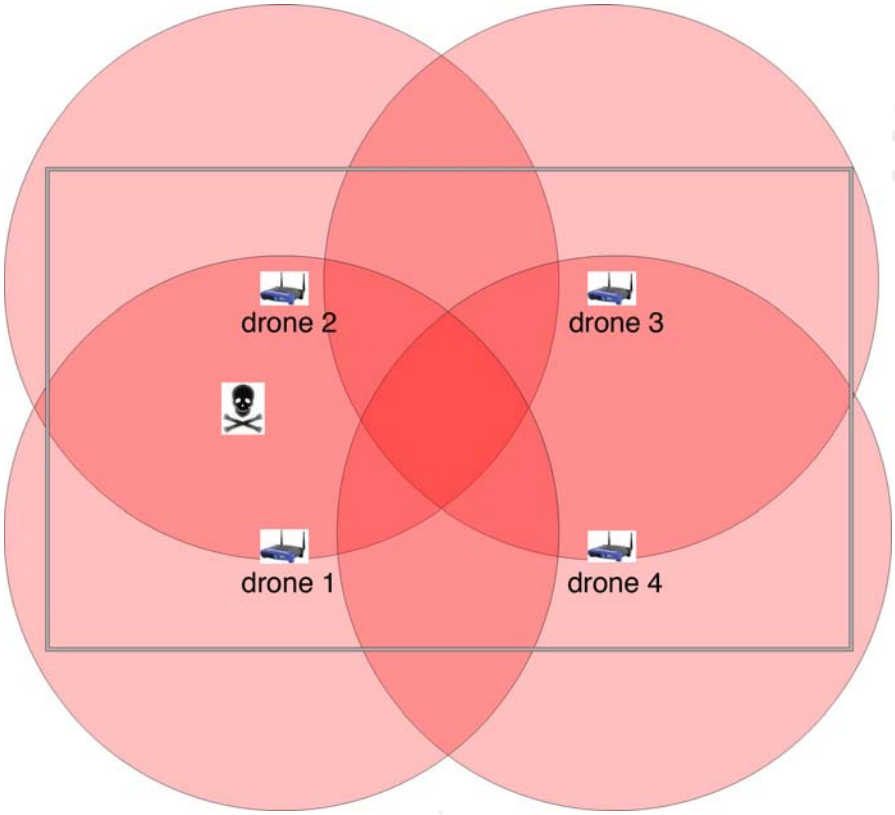
Figure 13 – Rogue in corner



It is possible for us to get lucky and find our rogue AP on our first drone that we cycle though, but this may not give us enough information to narrow down our rogue. It is a good idea to cycle through all of our drones in a particular geographic location; don't forget to not only think horizontally (on a single floor), but vertically as well (across multiple floors. If you were to install multiple drones across several floors, it may be possible for drones to discover a rogue AP from a floor above and below. By only checking one drone for our rogue, we could go on a wild goose chase attempting to recover our rogue.

It is also important to examine multiple drones in the same geographic location to narrow down search areas for recovery. For example, assume we have a rogue AP in the location as shown in Figure 14.

Figure 14 – Rogue in overlapping coverage area



In this instance, if we were to only check one drone, we'd probably want to begin our recovery in the areas covered by drone 1, which is a fairly significant area. If we had checked all of our drones in our geographic location we would have noted that the rogue was actually seen by drone 1 and drone 2 in the overlapping drone coverage area. We would have also noted that the rogue AP was not seen by drone 3 and drone 4, so these areas could be eliminated from the start. Given that this rogue was seen by two access points, it is a fairly good indication that it is located in the overlap zone for drone 1 and drone 2. This overlap area is significantly smaller than the location we could have searched had we only investigated drone 1. Depending on the size of the overlap area in an installation, we might have been able to skip the mapping for recovery section entirely and performed a manual search.

Once we have been able to narrow our search location by cycling through our drones, we will want to go mobile with our laptop and perform some mapping to recover the rogue AP.

Going Mobile and Notes About Antennas

In order for us to begin mapping the location of our drone we need to shut down Kismet (after recording the MAC address of the rogue!), and gather our equipment and head to the suspected location. When we arrive, we will want to insert our wireless card and attach our directional (yagi) antenna.

For this paper, the author elected to use a 300mW PCMCIA Atheros based wireless card from Ubiquiti[ix] available from a number of internet based retailers for about $130. Yes, in this case, the wireless card was more expensive than the disposable laptop! Additionally, a Senao[x] 200mW Prism 2.5 based card, available for about $60 could have been used for some more cost effectiveness. Both cards are illustrated in Figure 15.

Figure 15 – Senao (top) and Ubiquiti (bottom) PCMCIA network cards

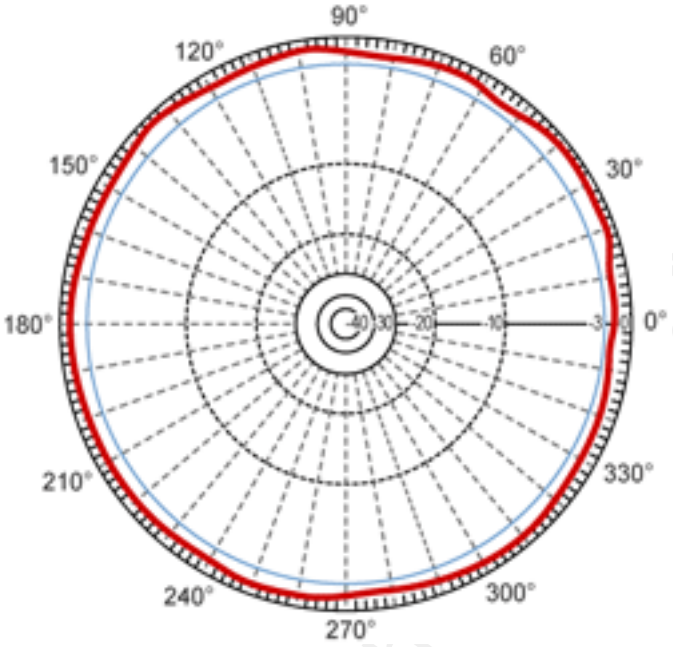
Additionally, a yagi antenna[xi] (Figure 17) and connector (pigtail)[xii] will be necessary to perform mapping for our rogues while mobile, and can be obtained for around $40. The antenna and pigtail can be interchanged between the two recommended cards. It may also be a good idea to order a pair of omni directional antennas[xiii] for either card, and be able to reuse the card for other purposes. These omni directional antennas can be purchased for about $15, and are also interchangeable between the two recommended cards. The omni antenna and yagi pigtail are shown in Figure 16.

Figure 16 – Yagi antenna

Figure 17 – Omnidirectional Antenna (left), Yagi pigtail (right)

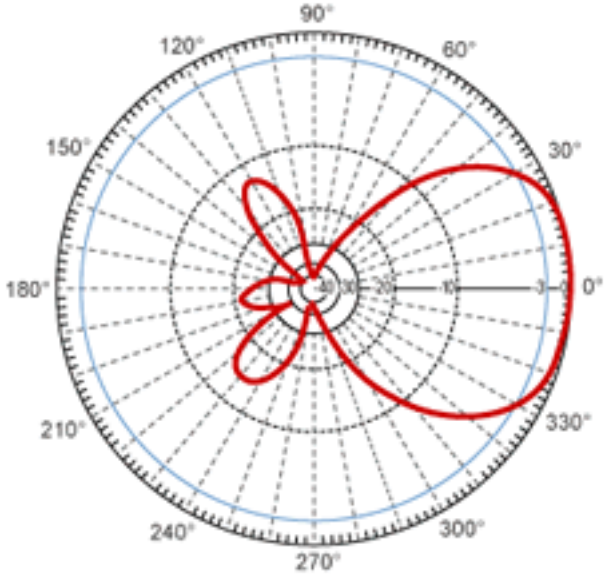
As you will note, we have just touched upon two different types of antennas; omni directional and yagi. These two antenna types have some very significant design differences for different tasks. The WRT54GL drones come equipped with omni directional antennas, which cover an area with wireless over a large, circular area shaped similar to a donut. The omni directional antennas typically are deployed horizontally, but can provide a "tall" coverage area that may extend through multiple floors in the same donut shape, as indicated by the coverage diagram in Figure 18.

Figure 18 – Omni antenna horizontal polarity coverage



A yagi antenna has a focused coverage area with a general shape of a slice of pie, with the point beginning at the antenna. Yagi antennas can have different widths of focus, depending on the design. In addition to the narrow focus, shaped like a pie, orientation of the antenna becomes very important as well. A yagi antenna is subject to polarization (orientation), in either the horizontal or vertical planes. Imagine that slice of pie laying on a plate (horizontal polarization) or standing on edge (vertical polarization). With that description, it is easy to understand how those features can be to our advantage. Note the "pie" shape of the yagi Figure 19.

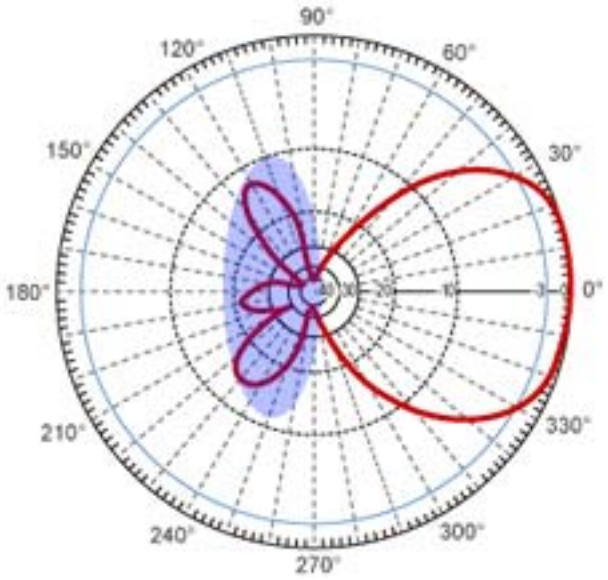
Figure 19 – Yagi antenna horizontal polarity coverage



With a yagi in horizontal polarization, we will be examining a pie shaped wedge on a single floor of our environment. With the vertical polarization, we'd be examining a narrow path over multiple floors. Seeing that we have cycled though our drones to determine what floor our rogue AP is on, vertical polarization would probably be overkill. For our best results for rogue AP recovery we will be using our yagi antenna with horizontal polarity.

One additional note on a yagi antenna is that while most of the direction of the antenna is facing "forward", there is a small bit of coverage behind the antenna as well as indicated in blue highlighting in Figure 20.

Figure 20 – Yagi antenna rear lobes



It is important to note this as we map our readings to recover rogue APs, as these rear lobes can detect a close rogue AP with a strong signal, and it is possible to have that rogue be located behind the reading location. We'll discuss these pitfalls more in depth later in this paper.

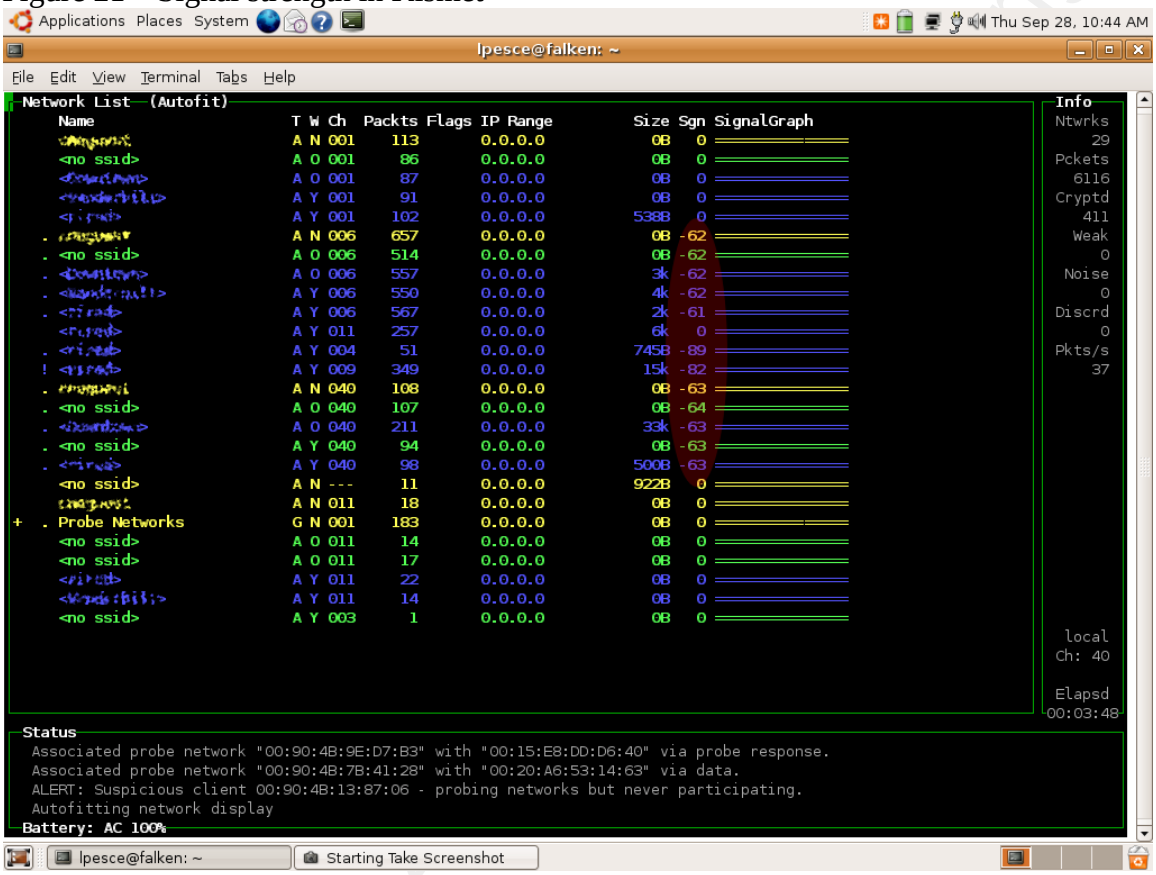
Now that we understand the differences in our antennas, we need to take our rogue discovery mobile. Shut down our multiple drone kismet monitor down after recording the rogue MAC address, insert our wireless card and attach our yagi antenna. We will then want to restart Kismet with our configuration for our mobile setup; the configuration that we created earlier with no drone capture sources and our wireless card as a local capture source. We can start our Kismet mobile setup by issuing the following command:

```
# sudo kismet –f /usr/local/etc/kismet_recover.conf
```

We should now notice that we have signal strength and signal to noise ratio (SNR) displayed in the Kismet display. Obviously, we are able to obtain signal strength from

the chipset in our PCMCIA wireless card, unlike that of the wireless chipset in the Linksys WRT45GL as we can see highlighted in red in Figure 21:

Figure 21 – Signal strength in Kismet



We have previously modified our kismet display to place this information on the main display, as this will be the vital information on performing our rogue AP recovery.

Mapping and Recovery

There are several methods that can be used for creating our maps for recovery; some electronic, some manual. Even with these two categories, there are many varied methods for performing the mapping and information gathering, all with their benefits and detractions. We will be illustrating the method that has proved the best results for the author while maintaining a low, disposable cost.

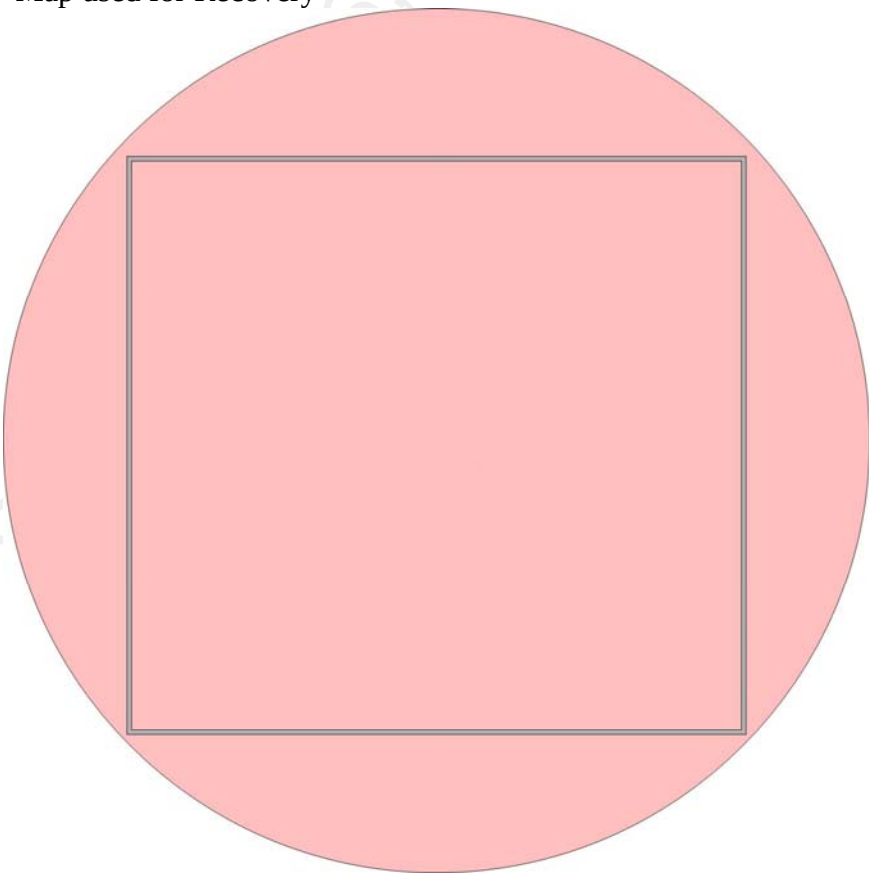
In the author's opinion, the best way to perform our mapping is the low-tech approach; paper, pencil and some educated guessing. With our low cost hardware, we certainly wouldn't want to do complex statistical analysis with our readings; it would probably take too much time. The paper and pencil can get us close enough, in a short amount of time.

The recovery process will almost never be able to give us an "X marks the spot" type of answer as to where a rogue AP is located. What our mapping process will give us is the ability to significantly reduce the area in which we will need to perform a physical search. A physical search will certainly be a part of the process, but by making some educated guesses through our mapping efforts, the area in which we have to search will be significantly less.

In addition to our paper and pencil, intuition is also a very integral part of the process. Unfortunately using signal strength and or signal to noise ratio (SNR) is not a perfect science, even in our perfect world scenarios that we have been using. In a real world example, there are a huge amount of outside forces that affect our signal strength and SNR; furniture, cordless phones, harmonic frequencies from unknown sources, building materials, microwaves, etc. The list can go on and on; with so many variables, educated guessing plays a large role. We'll take our readings and learn how to make some educated guesses.

As a first step, we will need to print off a copy of a map of the suspected area of our drone. In the event that a map is not available, draw up a reasonable representation of the area. Remember, it does not need to be perfect, or to scale! Find a suitable writing implement, and let us begin taking some readings. We'll use the map of our single drone environment for illustration purposes as indicated in Figure 22.

Figure 22 – Map used for Recovery

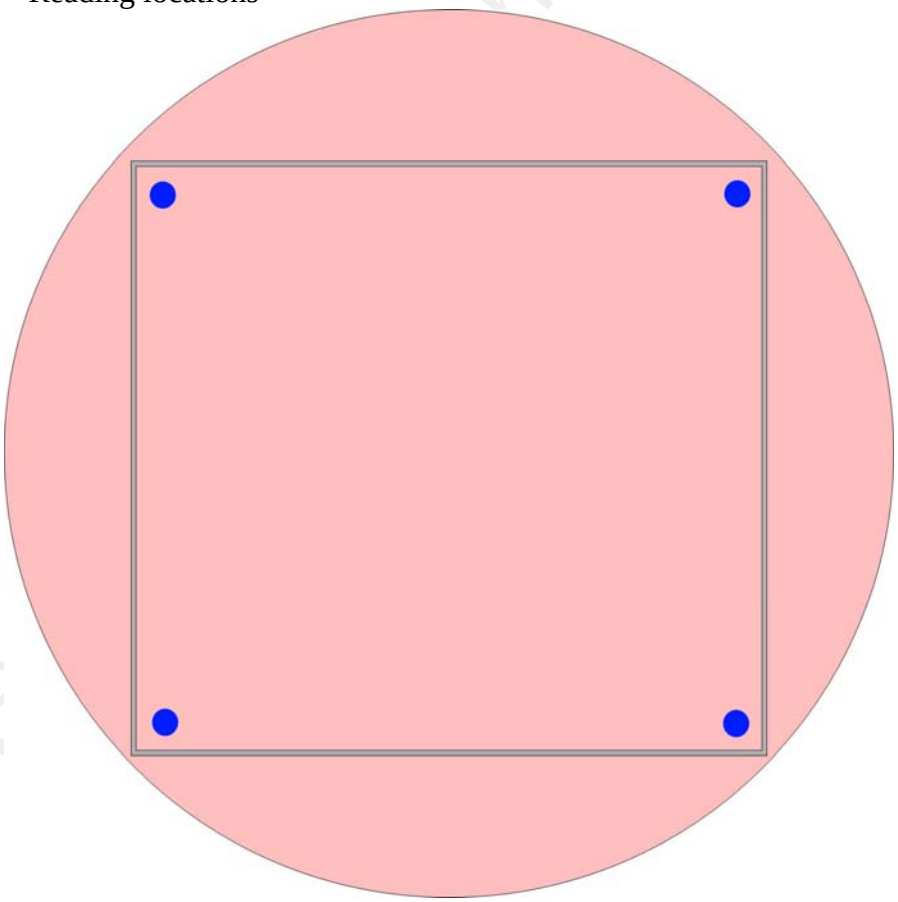


In this example, we'll assume that the rogue AP was only seen by a single drone so that we can illustrate how we will narrow down a large area for our physical search. The same concepts will apply to a smaller area or by multiple drones. Additionally, we've elected not to focus on a rogue AP that has been seen by two drones. In many real world cases, the area covered by two or more drones in which a rogue has been seen may be a small enough area to physically search without having to perform additional readings.

For this paper, the author will digitally mark up the example map for legibility purposes, where the real world example would be collected on paper. Certainly any real world example could be recorded digitally as well.

In order to begin mapping, we'll want to pick some common sense locations to take some signal readings from. Ideally we want to pick our reading locations form the far edges of the coverage area of the drone. In Figure 23, we've selected 4 locations, designated as blue circles at the edges of the drone #1 coverage area.

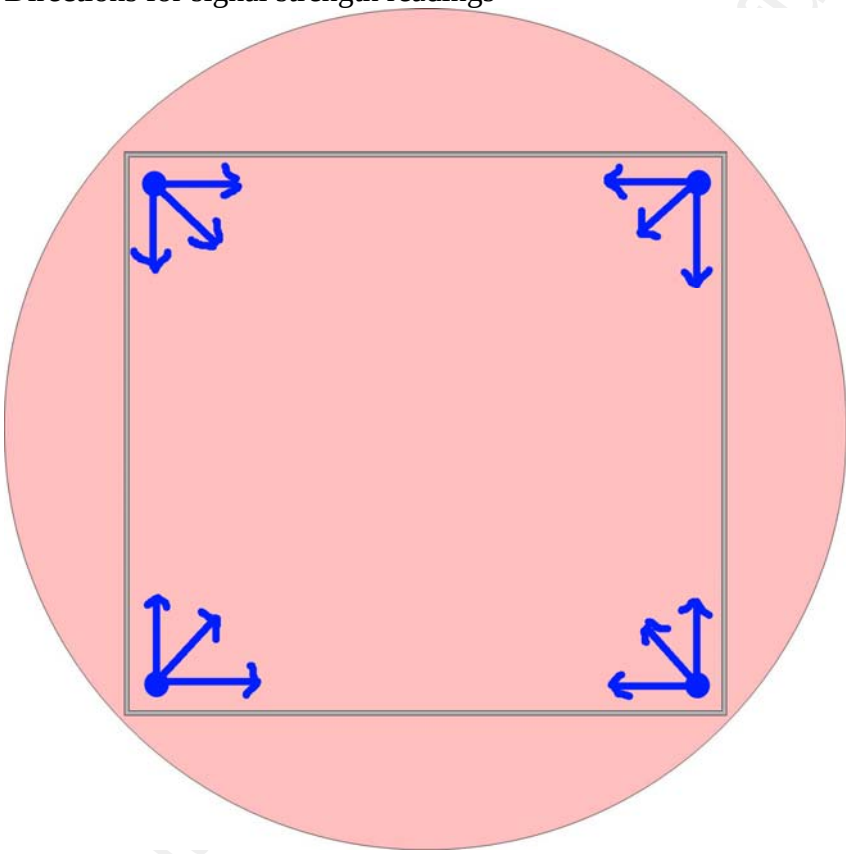
Figure 23 – Reading locations



From each of our test locations, we will need to take several readings with the yagi antenna pointed in different directions. We'll want to be sure that we are point the yagi inside our drone coverage area, as that is where the rogue would appear to be. You will

note that we will be taking multiple readings from each location. These directions are intended to cover the entire drone coverage area with our pie shaped yagi antenna coverage area. For each reading we should obtain enough information to to make educated guesses on the location of the rogue AP. In Figure 24 we will note the suggested directions for pointing the antenna for our readings.

Figure 24 – Directions for signal strength readings

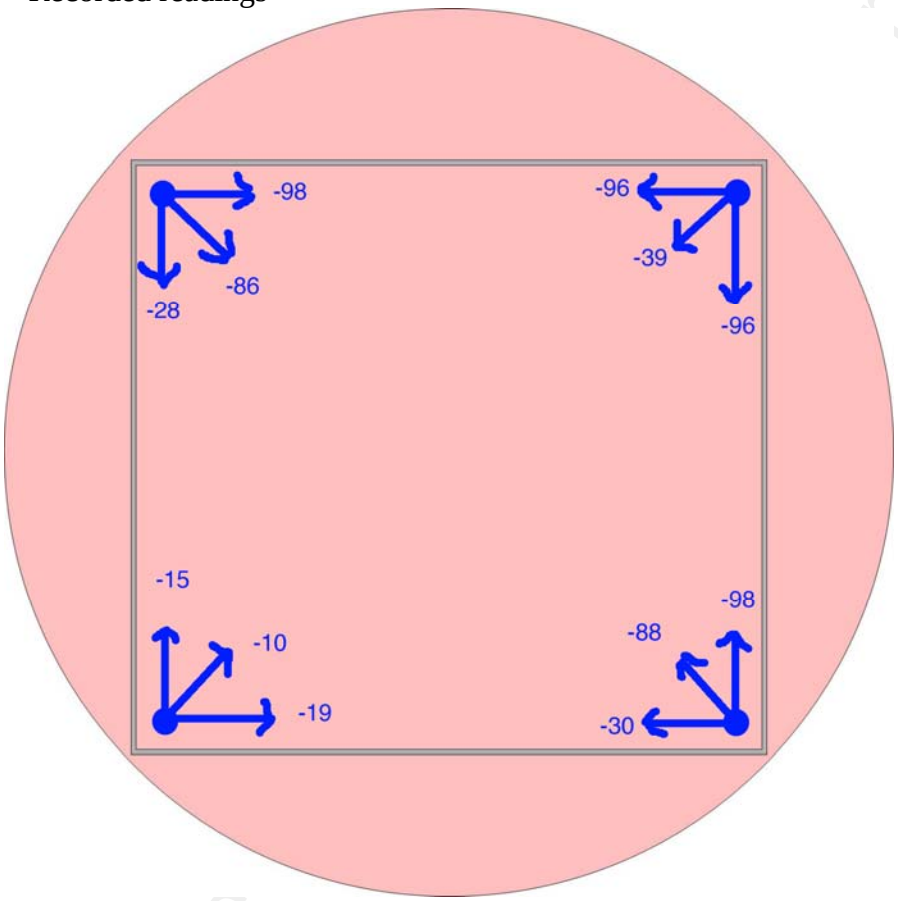


To obtain our readings, we will be using Kismet to record the signal strength of our rogue from each location and direction on our map. We'll be recording those values on the map for later interpretation. In order to obtain our readings, start Kismet with our recovery configuration file, sort the network list and locate the appropriate entry of the rogue AP by matching it to our previously recorded MAC addresses. These steps have been covered at length in this paper, so please refer to the appropriate sections for additional information.

For each location and direction, we will be recording the signal strength as indicated in the display of Kismet. We will note when we point the yagi and remain still for each direction that the signal strength will change each time the display will refresh. Obviously, this makes it difficult to take an absolutely accurate reading. We will probably notice that each direction will display dips and increases, however we will quickly notice that there will be a very distinct range that repeats. For our educated guess, take a quick mental average of several readings in the repeatable range during a 15-30 second time period, and record it on the map. We'll then need to repeat the same

signal strength reading process for all of our directions in all locations. When we are complete, we should have a map that looks similar to the map in Figure 25.

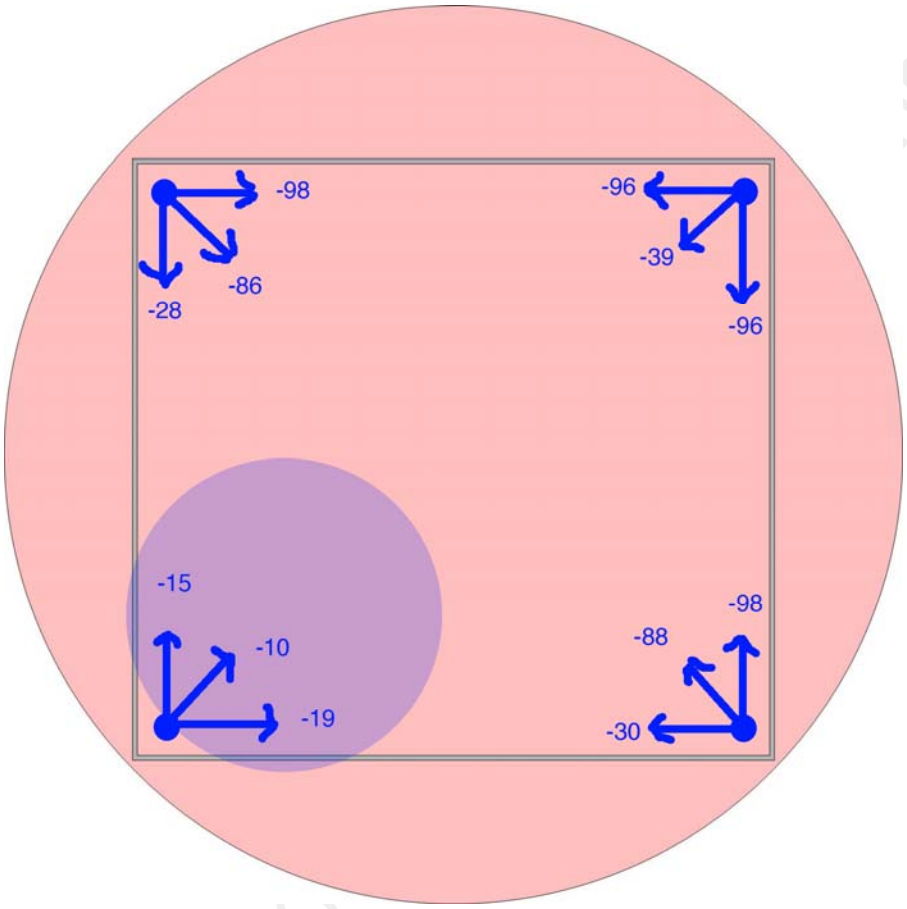
Figure 25 – Recorded readings



Once we have obtained all of our readings, we can begin to interpret them using some educated guesses. First off, we should note that with signal strength, bigger is better; the larger the number, the stronger the signal. We also need to remember some elementary math; -1 is actually larger than -5. So, a smaller negative number in our readings mean increased wireless signal.

By looking at our recorded figures, clearly the lower left hand corner has some readings that are significantly larger than the others. From that educated guess, we could be led to believe that the rogue AP is located fairly close to our reading location given that the signal strength is high. As a result, we'd probably want to focus our search area in that relative area. This estimated reduced search area has been indicated in Figure 26 in blue.

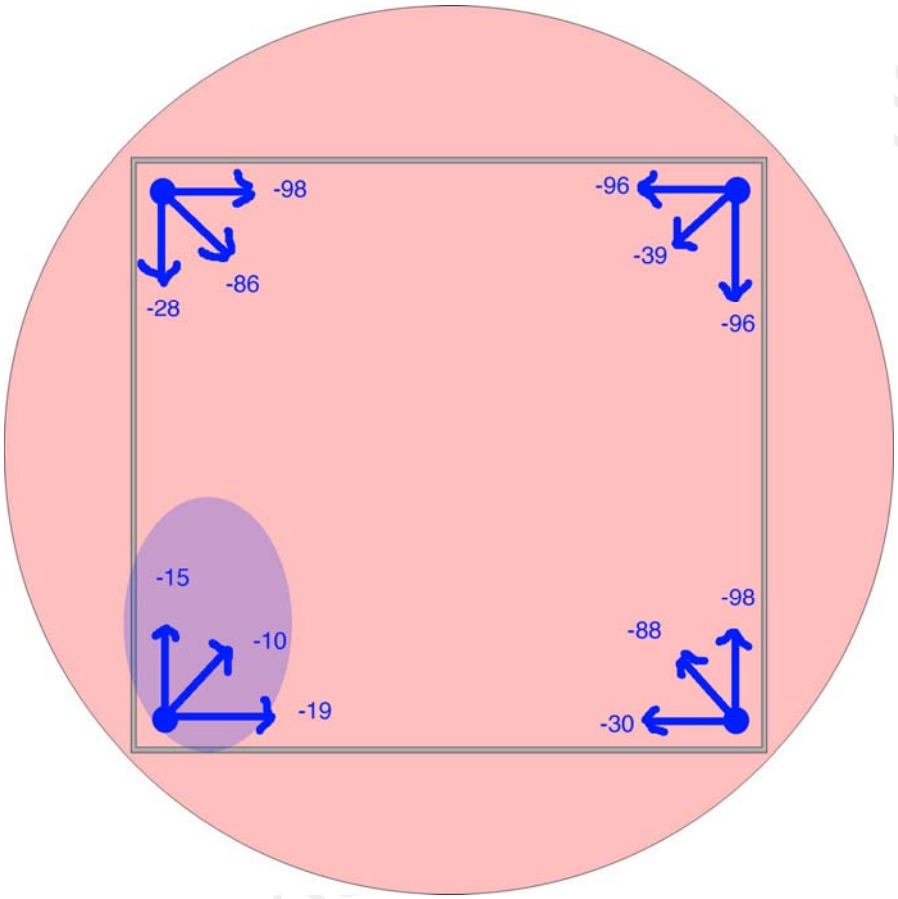
Figure 26 – Reduced search area



Additionally, we see some other supporting information from the three remaining read locations. The top left point has a reading of -28 pointing towards our reduced search location in the lower left. In the top right (-39) and in the bottom right (-30), we have additional, similar type of readings all pointing to our reduced search area. Clearly these particular readings are significantly different than the other readings (those in the -80s and -90s) from those same three locations.

In Figure 26, we have centrally located our reduced search area facing outward from our reading location. We can also attempt to shift that to the left or down, based on a more refined look at out readings. If we look at our readings in the lower left again, we can note that the signal readings would indicate that the rogue AP signal is stronger towards the -15 reading (pointing up) as opposed to the -19 reading (pointing right). If we then compare the stronger -15 reading and the opposing reading in the upper left corner of -28 to the weaker reading of -19 and the opposing reading in the lower right corner of -30. These comparisons do seem to indicate that the rogue AP would be located closer to the left hand side of our map. We can further refine our reduced search area as indicated in Figure 27.

Figure 27 – Refined reduced search area



We should be wary of our reduced search area in this case. Certainly these search areas are only a rule of thumb, but we should make note as to why we can be in error.

In this particular case we have indicated that the figures that we are relying on to refine our reduced area are indicative of a sifted search area, but we should note that the figures are very close. We have also indicated on our methodologies for reading that the signal strength can vary, and that we are only recording approximate values. As a result of these sampling methods, we are inserting a moderate margin of error. While many of the recorded signal level values have a significant difference, we do need to make a mental note about the relative margin of error. Through the understanding of this margin of error, we should understand that we should only use these refined search areas as a good starting point for our physical search. If in fact the rogue AP has not been located in our starting area, gradually expand the search area until the rogue AP has been located.

Now that we have defined a starting area for our physical recovery, we need to perform our physical search. We will want to look in all of the obvious places: on top of desks, in any equipment closets, etc. Don't forget about all of the not-so-obvious places: Behind large items of furniture, in cabinets and drawers, taped to the back side of a printer, etc. The possibilities are endless for hiding locations, especially for those determined to

disguise an AP. There have been reports of APs stuffed inside of a teddy bear[xiv], and inside of an apparently operational personal UPS device[xv].

A relatively fool proof method of locating rogue APs during a physical sweep would be to trace every network cable from the wall to final destination. This method will certainly locate any rogues that are easy to spot (such as a WRT54G, or a teddy bear), but could potentially overlook any devices such as a disguised AP (such as the UPS example). In the instance that there may be a disguised AP, it may be necessary to validate the functionality of every device in the search area.

Conclusion

The objective of this paper was to instruct the reader on how to install and operate Kismet and use disposable hardware to locate rogue APs. Along the way, we have discovered that there are a number of issues with this type of installation using disposable hardware.

Unfortunately, there are a number of issues obtaining any signal information from kismet_drone on a WRT54GL due to the nature of the wireless drivers. While this may change in the future, either by additional support with the existing drivers, or by the change of wireless chipsets in future WRT54G series of devices. It is likely that the chipset change will happen, as Linksys is releasing a WRT54G version 7, which is rumored to contain an Atheros wireless chipset, much like our Ubiquiti PCMCIA card.

Due to the lack of signal information, we have to reduce our initial search areas with multiple drones or by using Kismet on a very inexpensive laptop. Unfortunately this process is not very streamlined or terribly efficient, and as a result would not be very practical for a large, or geographically separated environment. It would be suitable for a mid-sized business looking to implement rogue AP detection with a small capital cost. This type of installation may also work as an interim, temporary, or proof of concept situation in order to justify a larger, more efficient installation.

Using Kismet, kismet_drone and disposable hardware can be an effective solution at low cost, but there are a number of technical difficulties associated with this type of installation.

---

[i] From http://www.kismetwireless.net
[ii] From http://www.linksys.com/servlet/Satellite?c=L_Product_C2&childpagename=US%2FLayout&cid=1133202177241&pagename=Linksys%2FCommon%2FVisitorWrapper
[iii] From http://www.kismetwireless.net/documentation.shtml#readme
[iv] This script was originally written by Joshua Wright, and can be found at http://www.renderlab.net/projects/wrt54g/S70JW_scan
[v] http://www.renderlab.net
[vi] The particular channel hopping pattern was determined by Joshua Wright, in currently undocumented research. This channel hopping pattern is the best option to eliminate any

bleed through for adjacent channels, as 802.11b channels overlap. Design considerations is support of Joshua's research can be found in an article at http://www.commsdesign.com/design_corner/showArticle.jhtml?articleID=16505876 by Greg Ratzel of Cirronet, Inc.

[vii] A detailed description of the Ubiquiti PCMCIA card can be found at http://www.ubnt.com/super_range_cardbus.php4

[viii] http://www.pauldotcom.com/KarmaUbuntu.pdf

[ix] http://www.ubnt.com

[x] A detailed description of the Senao PCMCIA card can be found at http://www.netgate.com/product_info.php?products_id=43

[xi] A detailed description of the Yagi antenna can be found at http://www.netgate.com/product_info.php?cPath=23_36&products_id=76

[xii] A detailed description of the MMCX pigtail can be found at http://www.netgate.com/product_info.php?cPath=21&products_id=30

[xiii] A detailed description of the MMCX omni-directional antenna card can be found at http://www.netgate.com/product_info.php?cPath=23_33&products_id=34

[xiv] A detailed write up of a teddy bear AP can be found at http://www.renderlab.net/projects/teddy-net/

[xv] As described in Renderman's presentation at DEFCON 14, indicated in slides 15-21 located at http://www.churchofwifi.org/FileLib_Index.asp?FID=27&LibName=DC%20%2014%20presentation%20material&PID=98