Global Information Assurance Certification Paper

An Inexpensive Wireless IDS using Kismet and OpenWRT

GAWN Gold Certification

Author: Jason Murray, jmurray@disillusion.ca

Adviser: Rick Wanner

Accepted: April 15, 2009

Table of Contents

1 Abstract

Wireless networks are everywhere these days. Enterprises large and small are setting up 802.11 wireless networks for numerous reasons including employee convenience, avoiding wiring costs, providing connectivity in warehouses, and courtesy access for guests, to name a few. In many cases enterprises are allowing critical business applications on the wireless network. Long gone are the days when wireless networking was just a toy for home use, or a convenience for customers at Starbucks.

The discipline of network security has as one of its goals the protection of critical business network traffic. There are a number of preventative methods that can be employed to ensure that a network is designed well, but attackers will still attempt to exploit weaknesses to gain access to important business data and systems. A technique that was developed over 25 years ago, intrusion detection, has been useful in monitoring the network for suspicious, possibly malicious traffic and raising the alarm when it is found. Intrusion detection systems (IDS), and their related cousins, intrusion prevention systems (IPS), have found wide spread adoption and use in many enterprise networks, whether small, medium or large.

Traditional IDS/IPS solutions focus on layer 3 and above threats. This implicit trust of layer 1 and layer 2 is not unwarranted as to gain access to the physical cabling plant and access a data port you either have to compromise the physical security or be an employee. However, 802.11 wireless networking is a different proposition. Indeed a wireless network has all the same threats against it at layer 3 and above, but it is also exposed to a unique set of threats at layers 1 and 2. This is inherent in the technology as it makes use of the electromagnetic spectrum in the radio frequency (RF) range as the common medium over which stations connect to the network. This necessarily exposes layers 1 and 2 to anyone

who is within the RF range of the network. Traditional IDS/IPS systems don't monitor for these kinds of threats.

Capable IDS/IPS is available for 802.11 specific needs, notably from AirDefense (recently purchased by Motorola), Cisco, AirMagnet and Aruba, but deployment is prohibitive for small and medium enterprises for a number of reasons. Cost is one of the main factors. The investment needed in equipment alone is prohibitive let alone the staffing levels and training necessary to configure, monitor and respond to raised alarms. Because of this many small and medium enterprises will opt to forego these systems to the detriment of their wireless network security.

There exists a low cost option. For a minor investment in equipment, the combination of Kismet on OpenWRT can provide IDS capability. But is it a viable option for small and medium enterprises? Will small enterprises have staff with the technical ability to install, maintain and monitor this solution? Are the reported alarms going to be understood by typical small company staff?

This paper sets out to explore these questions. The capabilities of both OpenWRT and Kismet will be discussed. A detailed explanation of how to install OpenWRT on a Linksys WRT54G, including the installation and configuration of Kismet for use as an IDS will follow. A number of such configured devices will be deployed in the field to gain operation experience with them. That experience will allow for a discussion on the suitability and application of the OpenWRT & Kismet combination for small and medium enterprises. Lastly a look at future directions will provide some ideas for where the solution's weaknesses can be improved.

2 Background

The Linksys WRT54G is an inexpensive residential network device capable of operating as a wireless access point (AP) that communicates over both 802.11b and 802.11g. These devices are popular and have undergone a number of revisions. In January 2006 revision 5 was released, moving the firmware from Linux to VxWorks, and reducing the memory (both flash and RAM). This reduction in memory made the platform incapable of running 3rd party firmware. However Linksys has re-released revision 4 under the name WRT54GL (Wikipedia, 2009). However, lack of commercial availability does not present a problem because even if a pre-revision 5 can not be found at a local computer store, they are widely available on eBay, typical selling price is in the $50-$60 range. There are a number of different revisions of the WRT54G available and OpenWRT works with all the revisions of the WRT54G prior to revision 5. There are however slight differences in the necessary configuration needed for each revision and these will be noted where appropriate. The equipment used by the author is revision 1.1.

The WRT54G ships with a firmware that limits customization. It meets many of the basic needs for home networks, including acting as a dhcp server, dns forwarding, primitive firewalling, network address translation (NAT), and wireless configuration without security or with WEP, WPA and WPA2 (Cisco, 2009). However, there is no way with the standard firmware for a technologically savvy user to expand the use of the platform beyond what is provided out of the box.

OpenWRT is a Linux based embedded platform initially targeted at the Linksys WRTG54, but now targets many embedded wireless devices including equipment from Asus, D-Link, NetGear, Soekris, Viewsonic and of course Linksys. Basically any equipment with

adequate flash memory and RAM that uses the Infeon ADM5120, TI AR7, Intel IXP4xx series, or the Broadcom BCN63xx series, can typically run OpenWRT (OpenWRT, 2008). The functionality may be limited in some instances, depending on the equipment in the device.

Support for these other hardware platforms implies that the approach taken in this paper could be applied to other equipment, even that supporting other 802.11 modes such as 802.11a and 802.11n. This is not explored in this paper. Operation over 802.11b and 802.11g was seen as sufficient to prove whether the concept was sound or not.

While OpenWRT is a “stripped down” version of Linux, in the sense that it does not provide a complete desktop experience, it still provides much of the functionality expected from a modern Linux distribution. Space on embedded devices is a very real constraint, because of this OpenWRT uses BusyBox as the shell environment. BusyBox embeds many of the common command line tools so a distribution does not have to provide individual executables. This cuts down on the space requirements needed allowing OpenWRT to provide a near complete Linux experience on devices with limited space (Anderson, 2008).

For software that is not included as part of the standard OpenWRT image, OpenWRT also supports ipkg. Ipkg is a packaging system similar in nature to other better known solutions such as Red Hat’s .rpm and Debian’s .deb. A distributed package repository is available for ipkg packages. This is how Kismet, as well as some other supporting software, will be installed on the WRT54G.

Kismet is an open source 802.11 wireless network analyzer (Kershaw, 2008). It makes use of the underlying supported hardware to allow an analyst to discover information about the surrounding networks. Kismet is extremely configurable. It can be configured to hop all channels and to take a survey of the entire wireless environment, it can be configured to hop

a select number of channels to discover the range of a network of interest, or it can be configured to stay on one channel in order to collect traffic in a dedicated fashion. Kismet is completely passive in its operation unlike other tools such as NetStumbler or inSSIDer. Kismet's architecture is distributed. It has a server component and a client component that can be operated on different machines. There is also a drone component that can be used as a remote listening device sending traffic back to a central server. It is available for Linux, Mac OS X, and Windows. OpenWRT supports kismet on the WRT54G platform.

3 Existing Work

The best known work for getting Kismet up and running on OpenWRT is written by Renderman (Renderman, 2005). This work is somewhat dated, as can be inferred by the versions of OpenWRT and Kismet referenced, but the concepts still apply. Some small modifications to his instructions were needed. There are other guides similar to Renderman's (Nugroho, 2005) (Intoverflow, 2008) but they all refer to his and do not make any substantial additions to the concepts.

Because OpenWRT can be installed on other hardware there are some pages that provide instructions on how to get kismet operating as a drone on that hardware. One page in particular describes how to do this with the Ubiquiti NanoStation 2 (Intoverflow, 2008). It provides very complete instructions including how to compile and package OpenWRT rather than relying on the provided installation packages.

In 2005 there was an effort to combine Kismet's ability to capture packets with the Snort IDS. This was the now defunct Snort Wireless project (Lockhart, 2005). The pages for Snort Wireless are still available; however the last post is from 2005 so it can be presumed

the project is dead. Follow-up did not indicate that the work from this project was rolled into the Snort project. Attempt to contact the author of the Snort Wireless project was unsuccessful.

4 Architecture

Figure 1 - OpenWRT Kismet IDS Architecture provides an overview of the solution architecture.

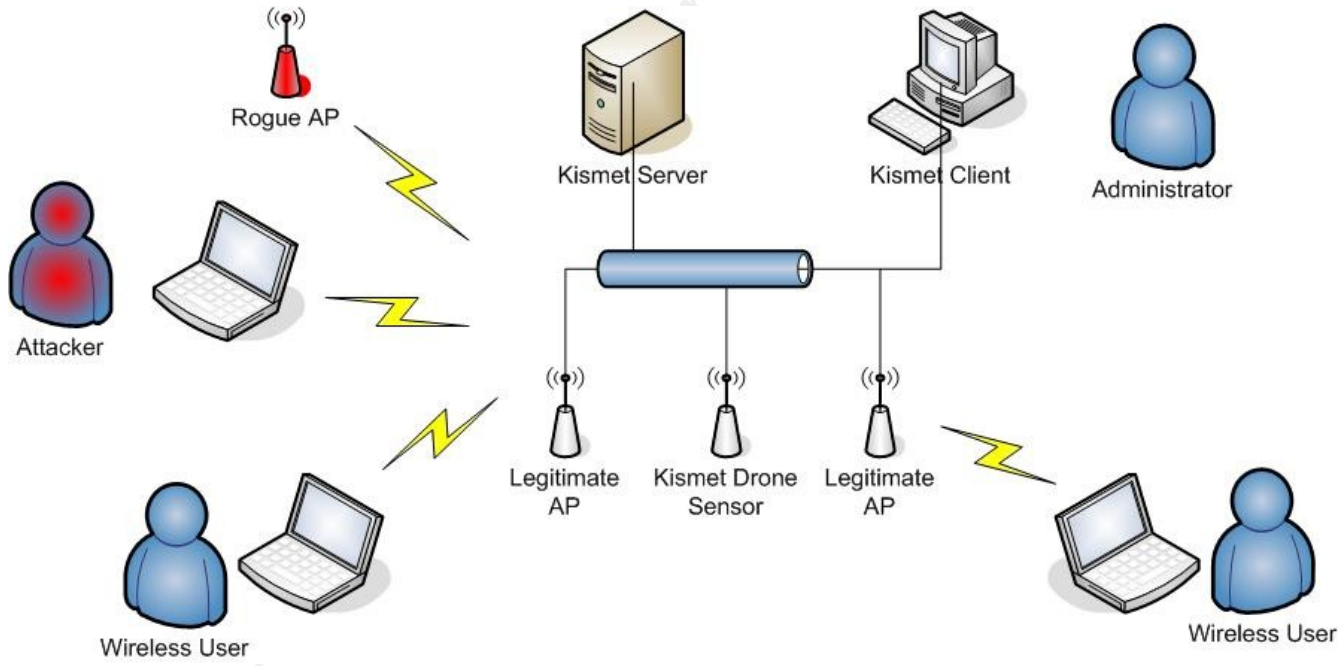

Figure 1 - OpenWRT Kismet IDS Architecture

The envisioned use is to provide a monitoring capability to small and medium enterprises that can not afford the wireless IDS offerings such as those from AirDefense and AirMagnet. The architecture will look much the same as those solutions, however. A network environment has some legitimate access points (AP) that are being used by wireless users.

Commercial solutions can either make use of existing “live” APs that handle user traffic by essentially stealing air time from legitimate clients, or you can deploy dedicated IDS sensors. In our solution there are some Kismet drone sensors placed throughout the environment. This increases the number of APs that ultimately need to be deployed, but the tradeoff is a dedicated IDS capability. It should be noted that the number of drones needed will be much less than the number of active APs because the drones are only listening to traffic and require much less power to do so.

The Kismet drone(s) are connected to the Kismet server and send wireless traffic over the network to the server. Using drones spread throughout the company’s location, all reporting back to one centralized server allows for good IDS coverage of the network and surrounding area, without having to monitor numerous individual devices.

The administrator uses the Kismet client to connect to the Kismet server to monitor the information captured from the Kismet drones in the field. This interactive use provides the typical Kismet user experience; allowing viewing of observed networks, the details of those networks, traffic captured from those networks, etc. It also provides for a centralized window of any alerts that Kismet detects. Due to limitations in the WRT54G hardware the normal ability to control channel locking and hopping is not available. Channel hopping is controlled locally on the Kismet drones.

Interactive use is not envisioned to be the normal use. The Kismet server can be run as a daemon and the alerting sent to the system logging services. Multilog, a component of the daemontools developed by Daniel J. Bernstein was chosen for its ease of use and familiarity on the part of the author. Multilog is used to capture all “ALERT” messages from the daemonized Kismet server process. Swatch (The Simple WATCHer of Logfiles) is the

used to further reduce and refine the logging that is captured allowing for meaningful alerting to be triggered for events of interest.

The end result is that the Administrator can be notified of any wireless events of interest (DoS, rogue APs, etc.) via email or SMS without needing to monitor an interactive Kismet client.

5 Concept of Operations

There are two primary threats that we are concerned with. The first being attacks from an actively malicious user; these could be a denial of service attack, active attempts to amplify traffic (arp replay) or break encryption (chop-chop). The second threat is that of rogue APs. These could be misconfigured APs connected to the company network, non-approved APs connected to the network by employees or attackers, or malicious APs not connected to the company network attempting to entice clients to associate preferentially with them.

The typical detection scenario proceeds as follows. An attacker or rogue AP will generate traffic over the air that the nearest Kismet drones will receive. It is possible that multiple drones will receive the malicious traffic. It is also possible that a drone will not detect the malicious traffic at all due to not monitoring the appropriate channel during the time of attack. This is because Kismet (and other commercial IDSs) hops from channel to channel listening for interesting traffic. While it is possible to listen to all channels concurrently this would require a device with one radio for each channel, this is cost prohibitive and no solution taking this approach is available commercially or otherwise. The hope is that as the drone hops channels it will eventually observe the malicious traffic. However it is possible that a stealthy (and lucky) attacker could guess the channel hopping pattern and completely avoid

being detected. This is not considered a likely possibility. The Kismet server will collect the traffic from the connected drones and detect the potential attack. An alert will be triggered and the administrator will be notified, whether via interactive console or automated monitoring. The alerts that Kismet generates for the different attack are covered elsewhere in this document.

This solution, unlike some commercial offerings, does not provide any active response to attacks or containment of rogue APs. The administrator will still have to track down the attacker or rogue AP and deal with the situation manually. While it is possible to add this capability to OpenWRT this was not undertaken for this paper.

6 Installation and Configuration

The documentation available on the OpenWRT site and elsewhere on the internet about performing installation and configuration of the WRT54G (and other hardware) with OpenWRT and Kismet is often out of date and consequently inaccurate. Additionally the documentation on the OpenWRT site covers both the previous (White Russian) and current (Kamikaze) release occasionally without any way to tell which version is being discussed. This section is an attempt to remedy that for the WRT54G rev 1.1 hardware when installing OpenWRT Kamikaze.

Procure a WRT54G (or other hardware supported by the OpenWRT project). It will likely be running the default Linksys firmware. It is possible that it might be running the DD-WRT 3rd party firmware.

Choosing the Right Firmware

In order to change the firmware on the WRT54G (or other hardware) it is necessary to download the appropriate firmware from the OpenWRT site. Determining which firmware to download requires checking with the Table of Hardware (OpenWRT, 2008) and determining which device and revision you have. Find the manufacturer and click through to the manufacturer page. On the manufacturer page (OpenWRT, 2008) find the specific product and click through to the details page in the very right hand column of the table, labeled "Status". (If you click on the product name in the left most column this will take you to the product's webpage and the manufacturer's site, not where you want to go.) In our case we have a Linksys WRT54G rev 1.1 (OpenWRT, 2008). The OpenWRT details page provides extensive information about the hardware including which OpenWRT release is supported, hardware specifications, how to set up serial port and JTAG connections, etc. What we are concerned with is discovering which vendor manufactured the wireless chip. Somewhere on the information page there will be a list of hardware specifications including the CPU architecture, vendor, speed, and the wireless chipset used. In our case the chip in use is the Broadcom BCM43xx. If you can not find the information on the OpenWRT website, Wikipedia often has the relevant information (Wikipedia, 2009)

With this piece of information you can then you can go to the download section of the OpenWRT website. This is accessed from the "Downloads" link on the front page of the OpenWRT website (OpenWRT, 2008). Choose the latest version, 8.09 as of this writing, and then select the version that matches the wireless chipset used by your hardware; in our case brcm-2.4. In this directory there will be a number of files available for download. One will be a .trx and the rest .bin. The .trx file is the firmware in "raw" format exactly as it will be written to flash. The various .bin files are simply the .trx file with a small header added to mark them

as valid upgrades for a given vendor. E.g. openwrt-brcm47xx-squashfs.trx is the raw firmware image file, and openwrt-wrt54g-squashfs.bin has a header added to allow the built-in Linksys firmware to recognize it as a valid upgrade. The OpenWRT site recommends using the .bin files only when the .trx file does not work (OpenWRT, 2008). It was found that in order to upload the .trx file to the WRT54G the extension of the firmware needed to be changed from .trx to .bin or the Linksys firmware would not allow it to be uploaded.

Flashing OpenWRT from the Linksys firmware

Connect your WRT54G to power and to the wired network. Then connect to the configuration interface with your web browser. The default URL is https://192.168.1.1 with a default blank username and a password of admin. If you are using a device that is not set to factory defaults enter the correct username and password to log in. You will be presented with a self-signed certificate warning. You will have to accept this and allow the connection to proceed. In Firefox (the author's preferred browser) you do this by adding a security exception. This exception can be temporarily allowed for only this session, as the new firmware will replace the certificate presented to the browser. Other browsers will have similar methods to accept the self-signed certificate and continue loading the page.

Figure 2 - Self Signed Certificate Warning

You will be presented with the following web page once you accept the self-signed certificate warning.

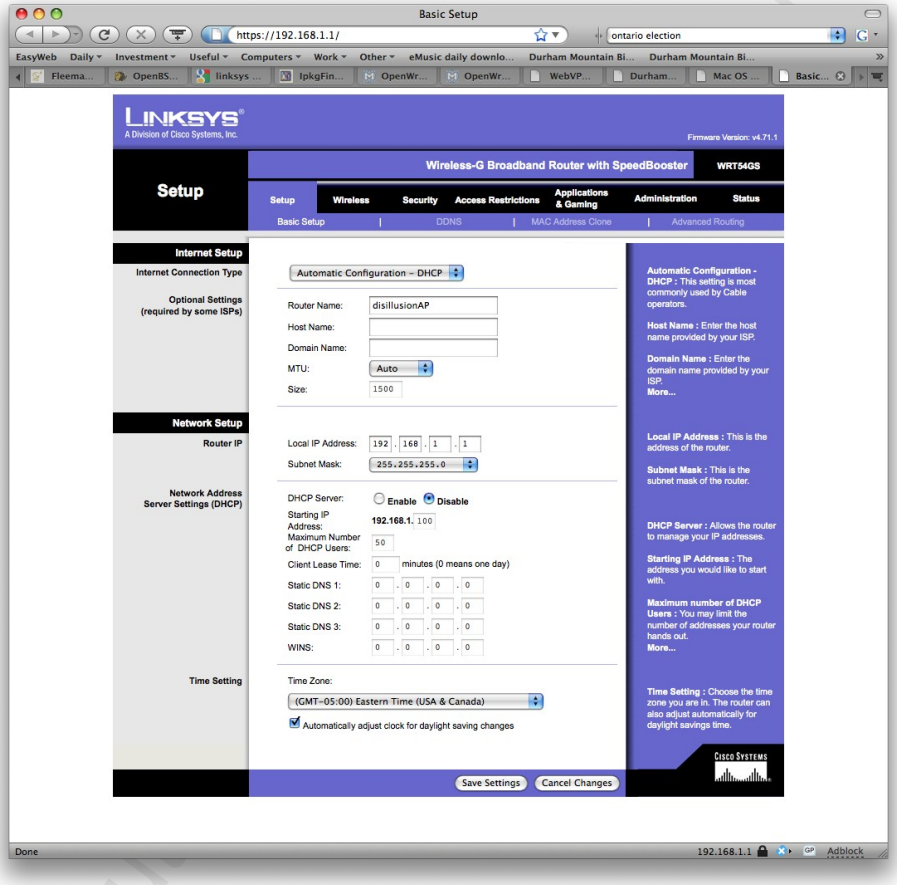

Figure 3 - Linksys Setup Page

From this page it will be necessary to navigate to the firmware upgrade page by clicking on the Administration tab (top right) then the Upgrade Firmware tab. This should present the following page.

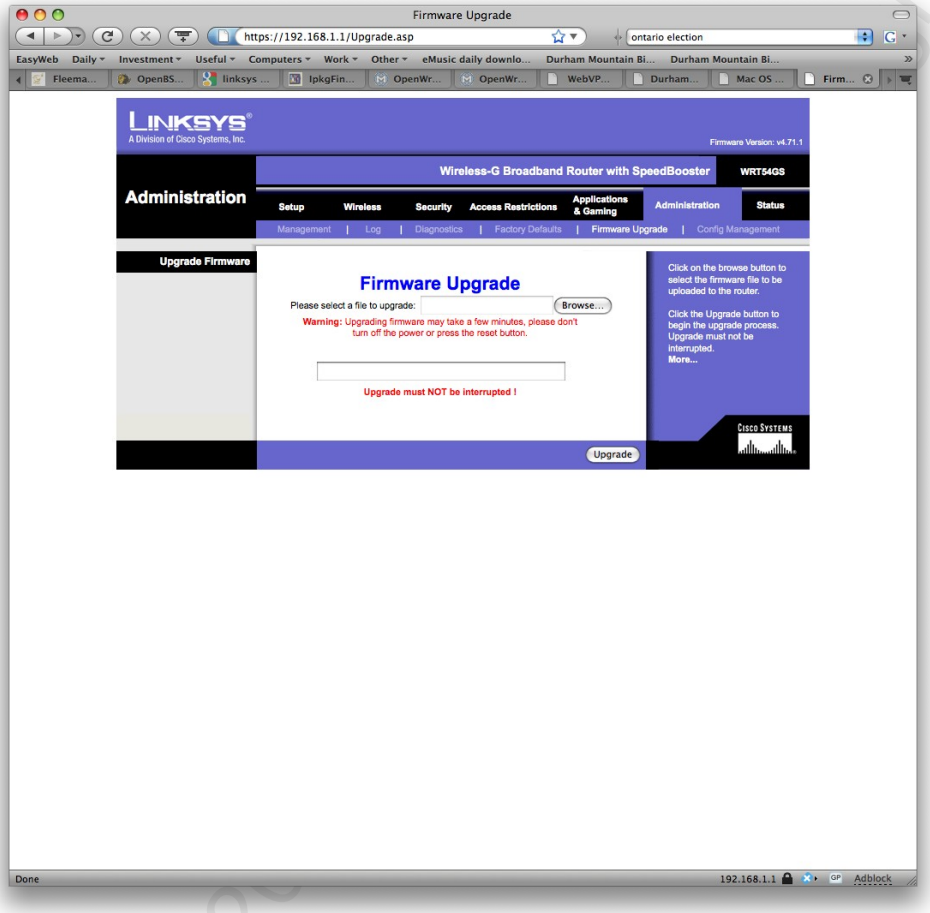

Figure 4 - Linksys Firmware Upgrade Page

Select the Browse button and select the firmware file previously downloaded.

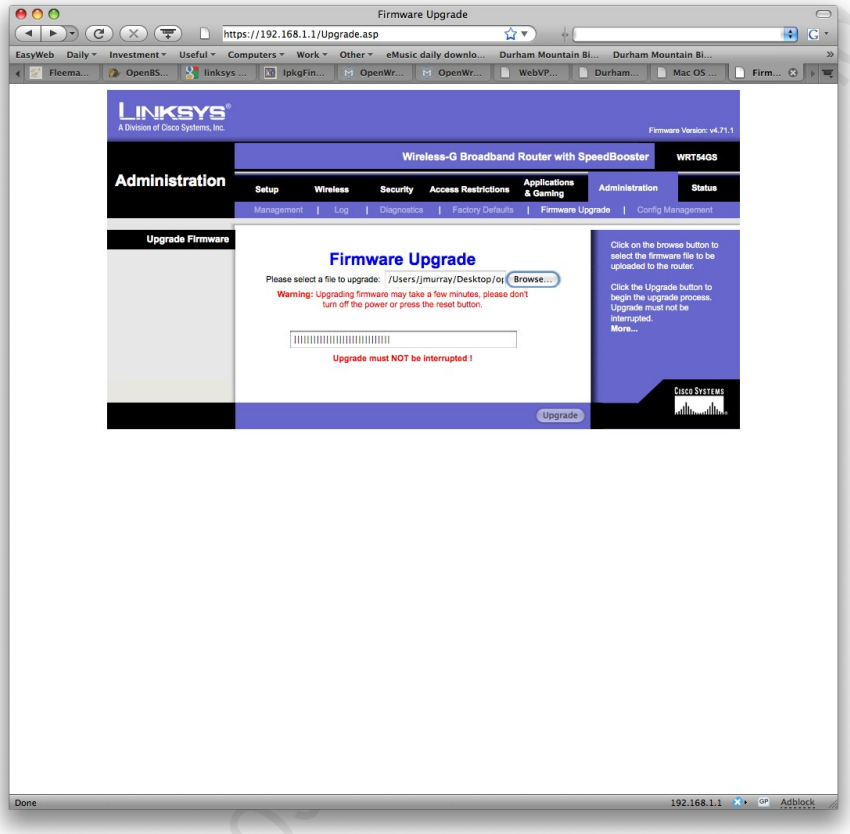

Figure 5 - Linksys Firmware Upload Progress Monitor

Once the firmware is uploaded and flashed the WRT54G will reboot into OpenWRT.

Connecting to OpenWRT

A freshly installed OpenWRT device will be listening for a login via telnet at the default IP address of 192.168.1.1. There will be no password set at this point (OpenWRT, 2008). Change the password with the `passwd` command. This will also disable telnet and enable ssh access to the device.

```
telnet 192.168.1.1
```

```
Trying 192.168.1.1...
Connected to 192.168.1.1.
Escape character is '^]'.
 === IMPORTANT ============================
  Use 'passwd' to set your login password
  this will disable telnet and enable SSH
 ------------------------------------------



BusyBox v1.00 (2006.03.24-09:16+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.


  _______                     ________        __
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
 WHITE RUSSIAN -----------------------------------
  * 2 oz Vodka   Mix the Vodka and Kahlua together
  * 1 oz Kahlua  over ice, then float the cream or
  * 1/2oz cream  milk on the top.
 ----------------------------------------------------
root@OpenWrt:~# passwd
Changing password for root
Enter the new password (minimum of 5, maximum of 8 characters)
Please use a combination of upper and lower case letters and numbers.
Enter new password:
Re-enter new password:
Password changed.
root@OpenWrt:~#
```

Once the password is set we can proceed to configure the network interfaces appropriately (OpenWRT, 2008). In our case we simply want to assign an IP address to the LAN interface as well as a default gateway and dns server. This configuration is stored in the `/etc/config/network` file. In the `config interface lan` section set `option ipaddr`, `option netmask`, `option gateway` and `option dns` appropriately.

```
root@kismet-drone1:~# cat /etc/config/network
#### VLAN configuration
config switch eth0
        option vlan0    "1 2 3 4 5*"
        option vlan1    "0 5"

#### Loopback configuration
```

```
config interface loopback
        option ifname   "lo"
        option proto    static
        option ipaddr   127.0.0.1
        option netmask  255.0.0.0

#### LAN configuration
config interface lan
        option type     bridge
        option ifname   "eth0.0"
        option proto    static
        option ipaddr   192.168.1.7
        option netmask  255.255.255.0
        option gateway  192.168.1.2
        option dns      192.168.1.2

#### WAN configuration
config interface        wan
        option ifname   "eth0.1"
        option proto    static
```

Restart the networking services with the following command: `/etc/init.d/network restart`. This will disconnect you from the WRT54G. You will have to reconnect this time using ssh. Make sure to use the –l option to specify the local user to login in as.

```
$ssh -l root 192.168.1.7
root@192.168.1.7's password:



BusyBox v1.4.2 (2007-09-29 09:01:24 CEST) Built-in shell (ash)
Enter 'help' for a list of built-in commands.


  _______                     ________        __
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
 KAMIKAZE (7.09) ------------------------------------
  * 10 oz Vodka       Shake well with ice and strain
  * 10 oz Triple sec  mixture into 10 shot glasses.
  * 10 oz lime juice  Salute!
 ---------------------------------------------------
root@kismet-drone1:~#
```

Installing and Configuring kismet_drone

Now that we have a working network configuration we can install the kismet_drone

package (as well as some supporting software). We will be using the publicly available ipkg repository. First we should update the repository files so we are sure to fetch and install the latest version of Kismet and wl (Renderman, 2005).

```
ipkg update
ipkg install kismet_drone wl
```

wl is a command line tool for managing the radio interface of the WRT54G it can make some tasks easier to perform than using `ifconfig/iwconfig` directly.

Kismet_drone will install its configuration files in `/etc/kismet`. We will make some modifications to `/etc/kismet/kismet_drone` for our setup. This includes allowing the Kismet server to connect to the drone, and which radio source kismet_drone should use. (Kershaw, 2008)

The IP address of the Kismet server must be permitted to connect. 192.168.1.2 is our kismet_server.

```
# People allowed to connect, comma seperated IP addresses or network/mask
# blocks.  Netmasks can be expressed as dotted quad (/255.255.255.0) or as
# numbers (/24)
allowedhosts=127.0.0.1,192.168.1.2
```

The correct source for rev 1.1 hardware is prism0 not eth1 or eth2 as indicated in the supplied kismet_drone.conf file and the Renderman documentation. (Renderman, 2005) (Harry66, 2008)

```
# To enable multiple sources, specify a source line for each and then use the
# enablesources line to enable them.  For example:
# source=prism2,wlan0,prism
# source=cisco,wl0,cisco
#source=wrt54g,eth1,wireless
# For v1 hardware uncomment this:
source=wrt54g,prism0,wireless
```

However there is a catch with the wireless source. It is not available immediately after

boot. Some extra work has to be done to force it to become available. This may seem excessive to force the prism0 source to be available but after testing it was determined that this represents the minimal number of commands needed. If hardware other than the Linksys WRT54G is used, it is likely that some experimentation will need to be done to discover what the wireless source is and if any special commands are needed to bring it online.

```
ifconfig wl0 up
iwconfig wl0 mode Monitor channel 1 txpower 5mW
ifconfig wl0 down
ifconfig wl0 up
iwconfig wl0 mode Monitor channel 1 txpower 5mW
```

With a valid and functioning wireless source we can start kismet_drone:

```
/usr/bin/kismet_drone -f /etc/kismet/kismet_drone.conf
```

Normally kismet will handle the channel hopping necessary for a wireless IDS to view the traffic on all channels. In the case of the chipset on the WRT54G kismet cannot do this (Renderman, 2005). So instead a script must be used to force the wireless radio to change channel on a periodic basis. Aside from having to reimplement one of kismet's features, there is one further shortcoming of this approach. Kismet by default will change channels five times a second, using a script we have to dwell on a channel for one second as the shell's `sleep` command's shortest interval is one second. The script is run once per minute from cron. The channel hopping script will move to a new channel each second visiting all channels in 11 seconds. It will do this 5 times in 55 seconds leaving 5 seconds for 5 more hops. Channels 1, 6, and 11 are hopped to again in these 5 seconds. While it is possible to develop a more sophisticated and balanced hopping algorithm it was felt that given most APs are using one of those channels by default this was a good compromise between simplicity and channel coverage. The full script is:

```
root@kismet-drone1:/etc/init.d# cat /usr/bin/freqhop.sh
#!/bin/sh /etc/rc.common
```

```
# Copyright 2009 Jason Murray

for i in  1 2 3 4 5 ; 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
# this will get us to 55 seconds
# so 5 more hops to get us to a minute
# unfortunately we stay on some channels more frequently

wl channel 1 ; sleep 1
wl channel 6 ; sleep 1
wl channel 11 ; sleep 1
wl channel 6 ; sleep 1
wl channel 11 ; sleep 1
```

As mentioned this is run from cron once per minute.

```
root@kismet-drone1:/etc/init.d# crontab -l
* * * * * /usr/bin/freqhop.sh
```

Installing and Configuring kismet_server

As discussed in Section 4 – Architecture the kismet_drones send their traffic to a centralized Kismet Server. Kismet_server is supplied as part of the Kismet package and is available for common versions of Linux, *BSD, Mac OS X, and even Windows. It is left as an exercise for the reader to determine the correct installation commands for their preferred server platform. The Author uses OpenBSD and Kismet is available as a pre-built package. To fetch and install Kismet on OpenBSD issue the following command:

```
pkg_add kismet
```

Using kismet_server with kismet_drones requires the use of a "kismet_drone" source in the configuration file rather than a more typical wireless driver source (Kershaw, 2008). The configuration files for Kismet on OpenBSD are in `/etc/kismet`. `Kismet.conf` is the file that contains the configuration of the sources. Edit the `kismet.conf` file and find the `source` section. Set the source line similar to the following:

```
# Sources are defined as:
# source=sourcetype,interface,name[,initialchannel]
# Source types and required drivers are listed in the README under the
# CAPTURE SOURCES section.
# The initial channel is optional, if hopping is not enabled it can be used
# to set the channel the interface listens on.
# YOU MUST CHANGE THIS TO BE THE SOURCE YOU WANT TO USE
# OpenBSD examples:
# a/b:          source=radiotap_bsd_ab,ath0,radiotap_bsd_ab
# b:            source=radiotap_bsd_b,ipw0,radiotap_bsd_b
# b (wi(4))     source=prism2_openbsd,wi0,prism2_openbsd
source=kismet_drone,192.168.1.7:3501,drone1
source=kismet_drone,192.168.1.8:3501,drone2
```

Replace 192.168.1.7 and 192.168.1.8 with the IP addresses of your kismet_drones. You are not limited to two drones, you can have as many as you'd like, just be sure to place each drone source on a separate line. Ensure that the kismet_drones are online and allowing connections from our server or the server will not start up because it cannot connect to all its sources.

Before we can start kismet_server we have to make sure that multilog is installed. Multilog is a component of Daniel J. Bernstien's daemon tools. Daemontools are not available as a package for OpenBSD, but may be for your preferred server OS. If they are not, daemontools is relatively easy to compile and install from source. (Bernstein, 2009)

Start kismet_server with the following command:

```
/usr/local/bin/kismet_server 2>&1 | multilog -* +ALERT*
      /var/log/kismet/alerts &
```

This will start only the kismet_server component and redirect both stdout and stderr to multilog. Multilog, given the options above, will only capture lines starting with ALERT and put them in the `current` file in the `/var/log/kismet/alerts` directory. Ensure that `/var/log/kismet/alerts` exists before starting the server. After a while the log files will grow large enough for multilog to rotate them. A sample directory listing is:

```
# ls -l /var/log/kismet/alerts/
total 1908
-rwxr--r--  1 root  _kismet  98052 Feb 18 07:19 @40000000499bfce50dc64c6c.s
-rwxr--r--  1 root  _kismet  98000 Feb 19 01:33 @40000000499cfd4d11e9af8c.s
-rwxr--r--  1 root  _kismet  98105 Feb 19 16:52 @40000000499dd4bc2a918f64.s
-rwxr--r--  1 root  _kismet  98049 Feb 20 00:36 @40000000499e417428f71d2c.s
-rwxr--r--  1 root  _kismet  98048 Feb 20 10:39 @40000000499eced40688cc54.s
-rwxr--r--  1 root  _kismet  98079 Feb 20 20:28 @40000000499f58d90245f66c.s
-rwxr--r--  1 root  _kismet  98022 Feb 21 07:38 @40000000499ff5b917601ba4.s
-rwxr--r--  1 root  _kismet  98066 Feb 21 16:10 @4000000049a06de5071c751c.s
-rwxr--r--  1 root  _kismet  98066 Feb 21 23:23 @4000000049a0d32e3b2d392c.s
-rw-r--r--  1 root  _kismet  92019 Feb 22 10:05 current
-rw-------  1 root  _kismet      0 Feb  5 09:04 lock
-rw-r--r--  1 root  _kismet      0 Feb  5 09:12 state
```

Below is a sample of content from `current` file.

```
# head /var/log/kismet/alerts/current
ALERT Sat Feb 21 23:24:39 2009 Suspicious client 00:21:5C:2E:72:9D - probing
   networks but never participating.
ALERT Sat Feb 21 23:24:57 2009 Suspicious client 00:1B:11:C1:65:5D - probing
   networks but never participating.
ALERT Sat Feb 21 23:25:01 2009 Suspicious client 00:1F:3C:A2:DD:4A - probing
   networks but never participating.
ALERT Sat Feb 21 23:25:09 2009 Suspicious client 00:14:A5:DA:DC:F2 - probing
   networks but never participating.
ALERT Sat Feb 21 23:26:01 2009 Suspicious client 00:11:F5:15:FF:C6 - probing
   networks but never participating.
ALERT Sat Feb 21 23:26:13 2009 Suspicious client 00:13:E8:48:BE:1B - probing
   networks but never participating.
ALERT Sat Feb 21 23:26:50 2009 Suspicious client 00:21:5C:2E:72:9D - probing
   networks but never participating.
ALERT Sat Feb 21 23:26:53 2009 Suspicious client 00:1B:11:C1:65:5D - probing
   networks but never participating.
```

The “probing networks but never participating” ALERT is a common one and not very

useful, this will be discussed further in section 7 Operating.

At this point the system is capturing all the issues that Kismet ALERTS on. This information can be found in the Kismet Readme file (Kershaw, 2008). This logging information can be fed to any of a number of log monitoring tools or security information management tools. We will use Swatch: The Simpler WATCHer as our log monitoring tool (SWATCH, 2007). Swatch has a long history in the computer security field. Swatch is highly configurable both in what it can watch for and in how it can notify administrators when something is noticed. Various thresholds can bet set so that a notification will only be sent if a certain log message is seen X times in Y minutes. These thresholds are highly configurable and can be set to monitor a substring in a given log message. This way we can watch for a particular client or AP causing an ALERT rather than having to watch all ALERTs of a particular type. The high granularity of what to watch for is major benefit of using SWATCH. SWATCH also allows for a number of options in how notification can be sent. The log message can be echoed to the screen, sent to users if they are logged in to the machine, emailed to numerous people, and sent to another command for further processing (SWATCH, 2008).

Swatch is available as a package on most Linux and *BSD variants, including OpenBSD. It can be installed with the following command:

```
pkg_add swatch
```

The configuation file is for swatch is `~/.swatchrc` (~ representing the user's home directory) and is specific to the user invoking the swatch command rather than a system-wide configuration. This allows multiple users of a system to customize how they want SWATCH to behave. In our situation we will only be using one instance of SWATCH to watch the Kismet ALERT logs and notify the administrator of any interesting events. (Kershaw, 2008)

(SWATCH, 2008)

An excerpt from the `.swatchrc` used is show below:

```
watchfor /Suspicious client ([0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2}:
[0-9A-F]{2}:[0-9A-F]{2})/
    threshold track_by=$1,type=both,count=5,seconds=13200
    mail "nobody@example.com",subject="Suspicious Client"

watchfor /Out-of-sequence BSS timestamp on ([0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]
{2}:[0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2})/
    threshold track_by=$1,type=both,count=1,seconds=300
    mail "nobody@example.com",subject="Potential AP Spoof"
```

This will watch for the “Suspicious client” ALERT and track threshold information based on the “suspicious” client MAC address. An email will be sent once for each “suspicious” client that is noticed 5 times in each 3 hours and 20 minute window. The threshold resets after this time and if the “suspicious” client is notice 5 more times another email is sent. It will also watch for “out of sequence BSS timestamps” tracking threshold counts based on the BSS MAC address. An alert will sent for each detected potential spoof if one out of sequence message is detected in a 5 minute window. The full `.swatchrc` contains configuration for all ALERT messages of interest and is included in the Appendix. The threshold counts and times are the main avenue for configuration.

Execute SWATCH from command line with the following:

```
swatch --config-file /root/.swatchrc --tail-file
   /var/log/kismet/alerts/current
```

Provide the full path to the `.swatchrc` file to the `-config-file` option, and the actual log file to watch to the `-tail-file` option.

At this point we have a fully installed, configured, and functioning wireless IDS system. However it is not configured to start on reboot of any of the devices. To do this many of the

commands listed above will have to be included in the startup scripts of the WRT54G devices and the Kismet Server platform. Details of the various scripts needed are provided in the Appendix.

7 Operating

The author deployed the system described above in his residence and operated it over the course of a few months. The particular system consisted of one kismet_drone and a kismet_server running on an OpenBSD 4.3 server. The kismet_drone was monitoring two 802.11 networks, one the author's home wireless network and the other a network used for detection of various attacks or otherwise used to force Kismet to issue specific ALERTs.

There are very few wireless networks near the author's house so the amount of "natural" traffic to test with was minimal. Nevertheless it was instructive to observe what ALERTs the "background" traffic would generate. The overwhelming majority of ALERTs were for "suspicious" clients probing but never participating. It is not immediately obvious how these probes represent a security issue. It is possible that a company with strict wireless policy might want to know which devices in their environment are probing for wireless networks so this function can be disabled. Alternatively this might be a way for a company to identify misconfigured workstations that should be associated with the company wireless network but are probing for other networks instead. A sample of this alert is below:

```
ALERT Mon Mar 23 12:45:56 2009 Suspicious client 00:1A:73:02:6D:99 - probing
   networks but never participating.
```

The IDS system has been configured so that only one ALERT of this type per day per client MAC is being sent to the administrator. This still generated an excessive amount of

alerts. Given the questionable value of this ALERT it is likely that this ALERT will be ignored during day to day operation. It is recommended that this alert be ignored unless a company needs to monitor if it has probing clients.

Other ALERTs that were occurring in the course of normal operation included:

```
ALERT Mon Mar 23 03:35:45 2009 Suspicious traffic on 00:19:D2:85:76:A3.  Data
   traffic within 10 seconds of disassociate.
ALERT Sun Mar 22 19:36:56 2009 Beacon on 00:13:10:A0:68:FE (d-wifi) for
   channel 6, network previously detected on channel 5
ALERT Sun Mar 22 22:12:55 2009 Out-of-sequence BSS timestamp on
   00:1A:70:FC:C3:4E - got 1920b, expected 177214 - this could indicate AP
   spoofing
```

The first indicates a potential denial of service condition. Under normal operation a client will not initiate traffic in a relatively short time frame after a disassociation. This is most likely to occur if the client is being forcibly disassociated under an attack scenario. This can allow the attacker to impersonate a valid client in network that is a wireless hotspot or otherwise operating without encryption. This mode of operation is increasing unlikely in today's wireless environments but is still possible in an environment with resource constrained equipment that can not be easily upgraded.

The second indicates that an AP previously seen on one channel has moved to another. This could be deliberate on the part of the network administrators, in which case the ALERT would be expected and could be disregarded. This ALERT could appear legitimately in a few scenarios. One is if a rogue AP is deployed and the attacker attempts to impersonate the MAC address of a legitimate AP. This is not a typical deployment scenario for a rogue AP. Another legitimate appearance of this ALERT could be if an attacker gains access to a legitimate AP and modifies the configuration to change the transmitting channel. Modern enterprise 802.11 equipment has the ability to change channel automatically when it detects

too much noise on its currently configured channel. This behaviour would trigger this alert as well, but the type of equipment that supports this type of feature comes from the major enterprise vendors such as Aruba and Cisco, and supports embedded IDS functionality. A company that can afford a system that has automatic RF management capability is not likely to be monitoring their wireless environment with the system proposed in this paper.

The third ALERT can indicate AP spoofing. The comments about the channel change ALERT apply to this one as well. While detecting a spoofed AP is of value it is not a typically observed attack scenario in today's wireless environments.

In order to more completely test the system's capabilities various past and current attacks were executed. The following attacks were run against both the author's home network and a demonstration network used specifically for attacking purposes.

- NetStumbler and inSSIDer
- Traffic injection to crack WEP with Aircrack-ng
- Chop-chop
- Brute force password guessing

In each case Kismet did not issue an alert.

Kismet does have an ALERT that can be issued against NetStumbler but only for versions 3.22, 3.23, and 3.30 (Kershaw, 2008). The latest version of Netstumbler, which is still quite old, is 0.4.0. (NetStumbler, 2007) Another newer Netstumbler-like tool called inSSIDer (MetaGeek, 2009) was tested as well and it did not generate any ALERTs either.

The traffic based attacks were not expected to generate any ALERTs but were tested in the interest of thoroughness. Kismet does not list any of these attacks as something that it

will detect.

8 Challenges and Issues

The main issues have to do with the lack of customizability and extensibility of the solution. Some components of the solution allow for excellent customizability of what to watch for, how to apply thresholds, who will be notified and how, etc. But the main part of an IDS that needs to be easily customizable and extensible are the attacks that will be caught. The attacks that it will notice, and hence the ALERTs that it will report to an administrator, are hard coded in the Kismet software source. In addition there are attacks that are not detected by the system since Kismet has not been extended to watch for these attacks. There was hope that Kismet's "newcore" might rectify these issues. But "newcore" is a refactoring of the existing Kismet code only, (Kershaw, 2008) it does not include any enhancement to the IDS functionality. The same ALERTs are watched for in "newcore" as in the old code base.

A further challenge with the system as it is currently implemented is that it will only detect attacks against 802.11b and g networks. No detection is provided for a or n networks. This is only a practical limitation though as there is nothing in the design concept preventing the choice of hardware that covers all 4 types of 802.11 networks: a, b, g and n.

9 Future Directions

As its core component this system relies heavily on the features and capabilities of Kismet. As mentioned in Section 8 Challenges and Issues this presents challenges in adapting to new attacks. In order to remedy this Kismet would need to be modified to allow it to be easily adapted to watch for additional attack types. This would allow for the addition of new and novel attacks to be incorporated as they were encountered in the field. These

modifications could be as simple as a set of patches to the existing source code modifying it to monitor and ALERT on new attacks. Or it could be as extensive as needing rearchitecting of core components of the Kismet software.

As Kismet is not likely to be rearchitected or patched for these enhancements anytime soon alternatives should be sought out. There may be a possibility to use the kismet_drone's ability to redirect captured packets. These captured packets, once collected by the kismet_server they could be handed off to an external IDS system, perhaps SNORT, for more complete inspection.

Beyond simply detecting attacks, the system could be expanded to also perform prevention and containment. For example in the event of a rogue AP being detected over the air, a wired side network scan could be started to confirm if it also exists on the company's network. If so then spoofed disassociate packets could be sent to affected clients and the rogue AP to effectively contain the damage until the AP can be physically located and removed.

10 Summary

The need for an inexpensive capable wireless IDS system cannot be overstated. Not all companies can afford, nor wish to purchase, the expensive, yet capable, IDS systems from enterprise network providers such as Cisco, AirDefense (Motorola), AirMagnet and Aruba. Yet these same companies can purchase and deploy inexpensive wireless networks using consumer grade hardware. The need exists to provide these same companies the capability to monitor their wireless networks for attacks.

In this paper we discussed the use of the Linksys 54GL router, OpenWRT, and Kismet

as just such an inexpensive wireless IDS system. OpenWRT is an open source 3rd party firmware for the Linksys 54GL (and other devices) that provides for extensive customization and enhancement of the router above and beyond the firmware it ships with. OpenWRT permits for the installation of Kismet on the Linksys 54GL. Kismet has a distributed architecture that can be used to deploy passive drones throughout a wireless network coverage area. They all connect back to a central server that can provide centralized packet capture, monitoring and logging of alerts. Additional software on the server turns this into a fully functioning, if rudimentary, IDS system for 802.11 wireless networks.

Installation is relatively straightforward but due to inadequate and fragmented documentation requires quite a bit of research and background knowledge. Section 6 Installation and Configuration is an attempt to remedy this by pulling the various pieces together in one location.

Operation of the system has shown this solution does perform as intended, as a distributed IDS that will monitor a wireless LAN for attacks against it. But unfortunately due to a number of factors the alerts generated cause the solution to be of less value than was hoped for. Furthermore the majority of alerts are for older denial of service attacks and the system cannot be updated easily to monitor for newer attacks.

In 1 Abstract this paper poses three questions. Is this system a viable option for small and medium enterprises? From a cost and availability perspective this system is certainly viable, however the information to be gained from the system makes it less useful than was originally hoped for.

Will such enterprises have staff with the necessary skills to install, configure and operate the solution? The installation is not trivial and a fair bit of knowledge will be needed to

get the system operational. This includes at a minimum intermediate level knowledge of Linux, and wireless networking. A junior system administrator or network engineer would most likely struggle to get this system operational. The technical skills required to deploy and operate this solution is reasonably high.

Are the reported alarms going to be understood by these staff? The ALERTs issued from Kismet require a fairly deep understanding of wireless networking and in particular wireless network security. Fortunately these ALERTs can be filtered through Swatch in order to provide more context and explanation before forwarding them on to the ultimate recipient of the ALERT messages. On the other hand the value of an IDS/IPS system is really only realized by technical staff that possess good technical skills. An enterprise with weak technical staff, even if they did understand the ALERTs, might not know how to follow-up on them.

While the final outcome of this project is less than expected there are promising future directions that could be taken. The distributed packet capturing ability of Kismet in combination with a 3rd party IDS can be explored to see if this might provide better IDS extensibility and customizability. Looking beyond just detection, prevention and containment abilities could be added to the solution. The extensibility of OpenWRT makes this possible.

11 References

Erik Anderson (2008). BusyBox: The Swiss Army Knife of Embedded Linux. Retrieved April 6, 2009, from the BusyBox Web site: http://www.busybox.net/

D. J. Bernstein (2009). The multilog program. Retrieved January 11, 2009, from cr.yp.to Web site: http://cr.yp.to/daemontools/multilog.html

Cisco (2009). WRT54GL Product Page. Retrieved April 6, 2009, from Linksys by Cisco Web site: http://www.linksysbycisco.com/CA/en/products/WRT54GL

Harry66 (Jan, 2008). WRT54GSv2 WLAN monitoring. Retrieved February 10, 2009, from OpenWRT Forums site: http://forum.openwrt.org/viewtopic.php?id=14039

Intoverflow (Nov 14, 2008). OpenWRT and Ubiquiti NanoStation 2. Retrieved January 20, 2009, from Integer Overflow Web site: http://intoverflow.wordpress.com/2008/11/14/openwrt-and-ubiquiti-nanostation-2/

Mike Kershaw (2008). Kismet. Retrieved January 21, 2009, from the Kismet Web site: http://www.kismetwireless.net/

Mike Kershaw (2008). Kismet README. Retrieved January 21, 2009, from the Kismet Web site: http://www.kismetwireless.net/documentation.shtml

Mike Kershaw (2008). Kismet Downloads. Retrieved January 21, 2009, from the Kismet Web site: http://www.kismetwireless.net/download.shtml

Andrew Lockhart (2005). Snort Wireless. Retrieved December 29, 2008, from Snort Wireless Web site: http://snort-wireless.org/

MetaGeek (2009). inSSIDer a Wi-Fi network scanner for Windows. Retrieved March 15, 2009, from Meta Geek Web site: http://www.metageek.net/products/inssider

NetStumber (2007). Wirless Networking Tool. Retrieved March 14, 2009, from the NetStumbler Web site: http://www.netstumbler.com/

Himawan Nugroho (Feb 23, 2005). Kismet on WRT54G the Easiest Way. Retrieved January

20, 2009, from Inevitable Web site: http://brokenpipes.blogspot.com/2005/02/kismet-on-wrt54g-easiest-way.html

OpenWRT (2008). Table of Hardweare. Retrieved December 2, 2008, from OpenWRT Web site: http://oldwiki.openwrt.org/TableOfHardware.html

OpenWRT (2008). Hardware/Linksys. Retrieved December 2, 2008, from OpenWRT Web site: http://oldwiki.openwrt.org/Hardware(2f)Linksys.html

OpenWRT (2008). Linksys WRT54G. Retrieved December 2, 2008, from OpenWRT Web site: http://oldwiki.openwrt.org/OpenWrtDocs(2f)Hardware(2f)Linksys(2f)WRT54G.html

OpenWRT (2008). Kamikaze 8.09 download directory. Retrieved December 3, 2008, from OpenWRT Web site: http://downloads.openwrt.org/kamikaze/8.09/

OpenWRT (2008). OpenWRT Installation. Retrieved December 2, 2008, from OpenWRT Web site: http://oldwiki.openwrt.org/OpenWrtDocs(2f)Installation.html

OpenWRT (2008). OpenWRT Using, Bootup. Retrieved December 4, 2008, from OpenWRT Web site: http://oldwiki.openwrt.org/OpenWrtDocs(2f)Using.html

OpenWRT (2008). OpenWRT Kamikaze Configuration. Retrieved December 4, 2008, from OpenWRT Web site: http://oldwiki.openwrt.org/OpenWrtDocs(2f)KamikazeConfiguration.html

Renderman (2005). The Renderlab: WRT54G Kismet Drone How-To V0.3.3. Retrieved December 1, 2008, from The Renderlab Web site: http://www.renderlab.net/projects/wrt54g/openwrt.html

SWATCH (2007). The Simple WATCHer of Logfiles. Retrieved January 11, 2009, from the Swatch Web site: http://swatch.sourceforge.net/

SWATCH (2008). The SWATCH Manpage.

Wikipedia (2009). Linksys WRT54G Series. Retrieved April 8, 2009, from the Wikipedia Web site: http://en.wikipedia.org/wiki/Linksys_WRT54G_series

12 Appendix – Start-up and Configuration Files

In order for the system to start up automatically a number of configuration files and start-up scripts need to be in place, both on WRT54 (OpenWRT) and the Kismet Server (OpenBSD).

OpenWRT Start-Up and Configuration Files

Kismet Drone Start-Up: /etc/init.d/kismet_drone

```
#!/bin/sh /etc/rc.common
# Copyright 2009 Jason Murray

START=70
STOP=20

start() {

  echo -n "Setting radio for kismet_drone"
  /sbin/ifconfig wl0 up ; /usr/sbin/iwconfig wl0 mode Monitor channel 1 txpower 5mW
  /sbin/ifconfig wl0 down; /sbin/ifconfig wl0 up ; /usr/sbin/iwconfig wl0 mode Mon-
itor channel 1 txpower 5mW
  /usr/sbin/wl ap 0
  echo "."
  echo -n "Running kismet_drone"
  /usr/bin/kismet_drone -f /etc/kismet/kismet_drone.conf > /dev/null 2>&1 &
  sleep 3
  echo "."

}

stop() {

  killall kismet_drone

}

boot() {

# nothing special needs to be called on boot so just do start()
  start
}
```

Enabling Kismet Drone Start-Up. This will automatically start and stop the kismet_drone.

```
/etc/init.d/kismet_drone enable
```

Kismet Drone Configuration: /etc/kismet/kismet.conf

```
# Kismet drone config file

version=2005.04.R1

# Name of server (Purely for organiational purposes)
servername=Kismet

# User to setid to (should be your normal user)
suiduser=nobody

# Port to serve packet data... This probably shouldn't be the same as the port
# you configured kismet_server for, or else you'll have problems running them
# on the same system.
tcpport=3501
# People allowed to connect, comma seperated IP addresses or network/mask
# blocks.  Netmasks can be expressed as dotted quad (/255.255.255.0) or as
# numbers (/24)
allowedhosts=127.0.0.1,192.168.1.2
# Maximum number of concurrent stream attachments
maxclients=5

# Packet sources:
# source=capture_cardtype,capture_interface,capture_name
# Card type - Specifies the type of device. It can be one of:
#     cisco         - Cisco card with Linux Kernel drivers
#     cisco_cvs     - Cisco card with CVS Linux drivers
#     cisco_bsd     - Cisco on *BSD
#     prism2        - Prism2 using wlan-ng drivers with pcap support (all
#                     current versions support pcap)
#     prism2_hostap - Prism2 using hostap drivers
#     prism2_legacy - Prism2 using wlan-ng drivers without pcap support (0.1.9)
#     prism2_bsd    - Prism2 on *BSD
#     orinoco       - Orinoco cards using Snax's patched driers
#     generic       - Generic card with no specific support.  You will have
#                     to put this into monitor mode yourself!
#     wsp100        - WSP100 embedded remote sensor.
#     wtapfile      - Saved file of packets readable by libwiretap
#     ar5k          - ar5k 802.11a using the vt_ar5k drivers
# Capture interface - Specifies the network interface Kismet will watch for
#  packets to come in on.  Typically "ethX" or "wlanX".  For the WSP100 capture
```

```
#  engine, the WSP100 device sends packets via a UDP stream, so the capture
#  interface should be in the form of host:port where 'host' is the WSP100 and
#  'port' is the local UDP port that it will send data to.
# Capture Name      - The name Kismet uses for this capture source.  This is the
#   name used to specify what sources to enable.
#
# To enable multiple sources, specify a source line for each and then use the
# enablesources line to enable them.  For example:
# source=prism2,wlan0,prism
# source=cisco,wl0,cisco
#source=wrt54g,eth1,wireless
# For v1 hardware uncomment this:
source=wrt54g,prism0,wireless

# Comma-separated list of sources to enable.  This is only needed if you wish
# to selectively enable multiple sources.
# enablesources=prism,cisco

# Do we channelhop?
channelhop=true

# How many channels per second do we hop?  (1-10)
channelvelocity=5

# By setting the dwell time for channel hopping we override the channelvelocity
# setting above and dwell on each channel for the given number of seconds.
#channeldwell=10

# Do we split channels between cards on the same spectrum?  This means if
# multiple 802.11b capture sources are defined, they will be offset to cover
# the most possible spectrum at a given time.  This also controls splitting
# fine-tuned sourcechannels lines which cover multiple interfaces (see below)
splitchannels=true

# Basic channel hopping control:
# These define the channels the cards hop through for various frequency ranges
# supported by Kismet.   More finegrain control is available via the
# "sourcechannels" configuration option.
#
# Don't change the IEEE80211<x> identifiers or channel hopping won't work.

# Users outside the US might want to use this list:
# defaultchannels=IEEE80211b:1,7,13,2,8,3,14,9,4,10,5,11,6,12
defaultchannels=IEEE80211b:1,6,11,2,7,3,8,4,9,5,10

# 802.11g uses the same channels as 802.11b...
defaultchannels=IEEE80211g:1,6,11,2,7,3,8,4,9,5,10

# 802.11a channels are non-overlapping so sequential is fine.  You may want to
```

```
# adjust the list depending on the channels your card actually supports.
#
defaultchannels=IEEE80211a:36,40,44,48,52,56,60,64,100,104,108,112,116,120,124,128,
132,136,140,149,153,157,161,184,188,192,196,200,204,208,212,216
defaultchannels=IEEE80211a:36,40,44,48,52,56,60,64

# Combo cards like Atheros use both 'a' and 'b/g' channels.  Of course, you
# can also explicitly override a given source.  You can use the script
# extras/listchan.pl to extract all the channels your card supports.
defaultchannels=IEEE80211ab:1,6,11,2,7,3,8,4,9,5,10,36,40,44,48,52,56,60,64

# Fine-tuning channel hopping control:
# The sourcechannels option can be used to set the channel hopping for
# specific interfaces, and to control what interfaces share a list of
# channels for split hopping.  This can also be used to easily lock
# one card on a single channel while hopping with other cards.
# Any card without a sourcechannel definition will use the standard hopping
# list.
# sourcechannels=sourcename[,sourcename]:ch1,ch2,ch3,...chN

# ie, for us channels on the source 'prism2source' (same as normal channel
# hopping behavior):
# sourcechannels=prism2source:1,6,11,2,7,3,8,4,9,5,10

# Given two capture sources, "prism2a" and "prism2b", we want prism2a to stay
# on channel 6 and prism2b to hop normally.  By not setting a sourcechannels
# line for prism2b, it will use the standard hopping.
# sourcechannels=prism2a:6

# To assign the same custom hop channel to multiple sources, or to split the
# same custom hop channel over two sources (if splitchannels is true), list
# them all on the same sourcechannels line:
# sourcechannels=prism2a,prism2b,prism2c:1,6,11

# Where to put the pid file?
piddir=/var/run
```

Channel Hopping Script: Running every minute in cron.

```
#!/bin/sh /etc/rc.common
# Copyright 2009 Jason Murray

for i in  1 2 3 4 5 ; 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
# this will get us to 55 seconds
# so 5 more hops to get us to a minute
# unfortunately we stay on some channels more frequently


wl channel 1 ; sleep 1
wl channel 6 ; sleep 1
wl channel 11 ; sleep 1
wl channel 6 ; sleep 1
wl channel 11 ; sleep 1
```

Channel Hopping Script: add as a cron job

```
root@kismet-drone1:/etc/rc.d# crontab -l
* * * * * /usr/bin/freqhop.sh
```

OpenBSD Start-Up Commands and Configuration Files

Kismet Server Start-Up: add to /etc/rc.local

```
# Start kismet_server
echo -n " kismet_server"
/usr/local/bin/kismet_server 2>&1 | multilog -* +ALERT* /var/log/kismet/alerts &
```

Kismet Server Configuration: /etc/kismet/kismet.conf

```
# Kismet config file
# Most of the "static" configs have been moved to here -- the command line
# config was getting way too crowded and cryptic.  We want functionality,
# not continually reading --help!

# Version of Kismet config
version=2007.09.R1

# Name of server (Purely for organizational purposes)
servername=Kismet

# User to setid to (should be your normal user)
suiduser=_kismet

# Do we try to put networkmanager to sleep?  If you use NM, this is probably
```

```
# what you want to do, so that it will leave the interfaces alone while
# Kismet is using them.  This requires DBus support!
networkmanagersleep=true

# Sources are defined as:
# source=sourcetype,interface,name[,initialchannel]
# Source types and required drivers are listed in the README under the
# CAPTURE SOURCES section.
# The initial channel is optional, if hopping is not enabled it can be used
# to set the channel the interface listens on.
# YOU MUST CHANGE THIS TO BE THE SOURCE YOU WANT TO USE
# OpenBSD examples:
# a/b:          source=radiotap_bsd_ab,ath0,radiotap_bsd_ab
# b:            source=radiotap_bsd_b,ipw0,radiotap_bsd_b
# b (wi(4))     source=prism2_openbsd,wi0,prism2_openbsd
source=kismet_drone,192.168.1.7:3501,drone1
#source=kismet_drone,192.168.1.8:3501,drone2

# Comma-separated list of sources to enable.  This is only needed if you defined
# multiple sources and only want to enable some of them.  By default, all defined
# sources are enabled.
# For example:
# enablesources=prismsource,ciscosource



# Automatically destroy VAPs on multi-vap interfaces (like madwifi-ng).
# Madwifi-ng doesn't work in rfmon when non-rfmon VAPs are present, however
# this is a fairly invasive change to the system so it CAN be disabled.  Expect
# things not to work in most cases if you do disable it, however.
vapdestroy=true



# Do we channelhop?
channelhop=true

# How many channels per second do we hop?  (1-10)
channelvelocity=5

# By setting the dwell time for channel hopping we override the channelvelocity
# setting above and dwell on each channel for the given number of seconds.
#channeldwell=10

# Do we split channels between cards on the same spectrum?  This means if
# multiple 802.11b capture sources are defined, they will be offset to cover
# the most possible spectrum at a given time.  This also controls splitting
# fine-tuned sourcechannels lines which cover multiple interfaces (see below)
channelsplit=true

# Basic channel hopping control:
```

```
# These define the channels the cards hop through for various frequency ranges
# supported by Kismet.   More finegrain control is available via the
# "sourcechannels" configuration option.
#
# Don't change the IEEE80211<x> identifiers or channel hopping won't work.

# Users outside the US might want to use this list:
# defaultchannels=IEEE80211b:1,7,13,2,8,3,14,9,4,10,5,11,6,12
defaultchannels=IEEE80211b:1,6,11,2,7,3,8,4,9,5,10

# 802.11g uses the same channels as 802.11b...
defaultchannels=IEEE80211g:1,6,11,2,7,3,8,4,9,5,10

# 802.11a channels are non-overlapping so sequential is fine.  You may want to
# adjust the list depending on the channels your card actually supports.
#
defaultchannels=IEEE80211a:36,40,44,48,52,56,60,64,100,104,108,112,116,120,124,128,
132,136,140,149,153,157,161,184,188,192,196,200,204,208,212,216
defaultchannels=IEEE80211a:36,40,44,48,52,56,60,64

# Combo cards like Atheros use both 'a' and 'b/g' channels.  Of course, you
# can also explicitly override a given source.  You can use the script
# extras/listchan.pl to extract all the channels your card supports.
defaultchannels=IEEE80211ab:1,6,11,2,7,3,8,4,9,5,10,36,40,44,48,52,56,60,64

# Fine-tuning channel hopping control:
# The sourcechannels option can be used to set the channel hopping for
# specific interfaces, and to control what interfaces share a list of
# channels for split hopping.  This can also be used to easily lock
# one card on a single channel while hopping with other cards.
# Any card without a sourcechannel definition will use the standard hopping
# list.
# sourcechannels=sourcename[,sourcename]:ch1,ch2,ch3,...chN

# ie, for us channels on the source 'prism2source' (same as normal channel
# hopping behavior):
# sourcechannels=prism2source:1,6,11,2,7,3,8,4,9,5,10

# Given two capture sources, "prism2a" and "prism2b", we want prism2a to stay
# on channel 6 and prism2b to hop normally.  By not setting a sourcechannels
# line for prism2b, it will use the standard hopping.
# sourcechannels=prism2a:6

# To assign the same custom hop channel to multiple sources, or to split the
# same custom hop channel over two sources (if splitchannels is true), list
# them all on the same sourcechannels line:
# sourcechannels=prism2a,prism2b,prism2c:1,6,11

# Port to serve GUI data
```

```
tcpport=2501
# People allowed to connect, comma seperated IP addresses or network/mask
# blocks.  Netmasks can be expressed as dotted quad (/255.255.255.0) or as
# numbers (/24)
allowedhosts=127.0.0.1,192.168.1.0/24
# Address to bind to.  Should be an address already configured already on
# this host, reverts to INADDR_ANY if specified incorrectly.
bindaddress=127.0.0.1,192.168.1.2
# Maximum number of concurrent GUI's
maxclients=5


# Do we have a GPS?
gps=false
# Host:port that GPSD is running on.  This can be localhost OR remote!
gpshost=localhost:2947
# Do we lock the mode?  This overrides coordinates of lock "0", which will
# generate some bad information until you get a GPS lock, but it will
# fix problems with GPS units with broken NMEA that report lock 0
gpsmodelock=false


# Packet filtering options:
# filter_tracker - Packets filtered from the tracker are not processed or
#                  recorded in any way.
# filter_dump    - Packets filtered at the dump level are tracked, displayed,
#                  and written to the csv/xml/network/etc files, but not
#                  recorded in the packet dump
# filter_export  - Controls what packets influence the exported CSV, network,
#                  xml, gps, etc files.
# All filtering options take arguments containing the type of address and
# addresses to be filtered.  Valid address types are 'ANY', 'BSSID',
# 'SOURCE', and 'DEST'.  Filtering can be inverted by the use of '!' before
# the address.  For example,
# filter_tracker=ANY(!00:00:DE:AD:BE:EF)
# has the same effect as the previous mac_filter config file option.
# filter_tracker=...
# filter_dump=...
# filter_export=...


# Alerts to be reported and the throttling rates.
# alert=name,throttle/unit,burst/unit
# The throttle/unit describes the number of alerts of this type that are
# sent per time unit.  Valid time units are second, minute, hour, and day.
# Burst rates control the number of packets sent at a time
# For example:
# alert=FOO,10/min,5/sec
# Would allow 5 alerts per second, and 10 alerts total per minute.
# A throttle rate of 0 disables throttling of the alert.
# See the README for a list of alert types.
alert=NETSTUMBLER,10/min,1/sec
alert=WELLENREITER,10/min,1/sec
alert=LUCENTTEST,10/min,1/sec
alert=DEAUTHFLOOD,10/min,2/sec
```

```
alert=BCASTDISCON,10/min,2/sec
alert=CHANCHANGE,5/min,1/sec
alert=AIRJACKSSID,5/min,1/sec
alert=PROBENOJOIN,10/min,1/sec
alert=DISASSOCTRAFFIC,10/min,1/sec
alert=NULLPROBERESP,10/min,1/sec
alert=BSSTIMESTAMP,10/min,1/sec
alert=MSFBCOMSSID,10/min,1/sec
alert=LONGSSID,10/min,1/sec
alert=MSFDLINKRATE,10/min,1/sec
alert=MSFNETGEARBEACON,10/min,1/sec
alert=DISCONCODEINVALID,10/min,1/sec
alert=DEAUTHCODEINVALID,10/min,1/sec

# Known WEP keys to decrypt, bssid,hexkey.  This is only for networks where
# the keys are already known, and it may impact throughput on slower hardware.
# Multiple wepkey lines may be used for multiple BSSIDs.
# wepkey=00:DE:AD:C0:DE:00,FEEDFACEDEADBEEF0102030405060708090 0

# Is transmission of the keys to the client allowed?  This may be a security
# risk for some.  If you disable this, you will not be able to query keys from
# a client.
allowkeytransmit=true

# How often (in seconds) do we write all our data files (0 to disable)
writeinterval=300

# How old (and inactive) does a network need to be before we expire it?
# This is really only good for limited ram environments where keeping a
# total log of all networks is problematic.  This is in seconds, and should
# be set to a large value like 12 or 24 hours.  This is intended for use
# on stationary systems like an IDS
# logexpiry=86400

# Do we limit the number of networks we log?  This is for low-ram situations
# when tracking everything could lead to the system falling down.  This
# should be combined with a sane logexpiry value to flush out very old
# inactive networks.  This is mainly for stationary systems like an IDS.
# limitnets=10000

# Do we track IVs?  this can help identify some attacks, but takes a LOT
# of memory to do so on a busy network.  If you have the RAM, by all
# means turn it on.
trackivs=false

# Do we use sound?
# Not to be confused with GUI sound parameter, this controls wether or not the
# server itself will play sound.  Primarily for headless or automated systems.
sound=false
# Path to sound player
```

```
soundplay=/usr/local/bin/play
# Optional parameters to pass to the player
# soundopts=--volume=.3
# New network found
sound_new=/usr/local/share/kismet/wav/new_network.wav
# Wepped new network
# sound_new_wep=${prefix}/com/kismet/wav/new_wep_network.wav
# Network traffic sound
sound_traffic=/usr/local/share/kismet/wav/traffic.wav
# Network junk traffic found
sound_junktraffic=/usr/local/share/kismet/wav/junk_traffic.wav
# GPS lock aquired sound
# sound_gpslock=/usr/local/share/kismet/wav/foo.wav
# GPS lock lost sound
# sound_gpslost=/usr/local/share/kismet/wav/bar.wav
# Alert sound
sound_alert=/usr/local/share/kismet/wav/alert.wav

# Does the server have speech? (Again, not to be confused with the GUI's speech)
speech=false
# Server's path to Festival
festival=/usr/bin/festival
# Are we using festival lite?  If so, set the above "festival" path to also
# point to the "flite" binary
flite=false
# Are we using Darwin speech?
darwinsay=false
# What voice do we use?  (Currently only valid on Darwin)
speech_voice=default
# How do we speak?  Valid options:
# speech    Normal speech
# nato      NATO spellings (alpha, bravo, charlie)
# spell     Spell the letters out (aye, bee, sea)
speech_type=nato
# speech_encrypted and speech_unencrypted - Speech templates
# Similar to the logtemplate option, this lets you customize the speech output.
# speech_encrypted is used for an encrypted network spoken string
# speech_unencrypted is used for an unencrypted network spoken string
#
# %b is replaced by the BSSID (MAC) of the network
# %s is replaced by the SSID (name) of the network
# %c is replaced by the CHANNEL of the network
# %r is replaced by the MAX RATE of the network
speech_encrypted=New network detected, s.s.i.d. %s, channel %c, network encrypted.
speech_unencrypted=New network detected, s.s.i.d. %s, channel %c, network open.

# Where do we get our manufacturer fingerprints from?  Assumed to be in the
# default config directory if an absolute path is not given.
ap_manuf=ap_manuf
client_manuf=client_manuf

# Use metric measurements in the output?
```

```
metric=true

# Do we write waypoints for gpsdrive to load?  Note:  This is NOT related to
# recent versions of GPSDrive's native support of Kismet.
waypoints=false
# GPSDrive waypoint file.  This WILL be truncated.
waypointdata=/var/log/kismet/way_kismet.txt
# Do we want ESSID or BSSID as the waypoint name ?
waypoint_essid=false

# How many alerts do we backlog for new clients?  Only change this if you have
# a -very- low memory system and need those extra bytes, or if you have a high
# memory system and a huge number of alert conditions.
alertbacklog=50

# File types to log, comma seperated
# dump    - raw packet dump
# network - plaintext detected networks
# csv     - plaintext detected networks in CSV format
# xml     - XML formatted network and cisco log
# weak    - weak packets (in airsnort format)
# cisco   - cisco equipment CDP broadcasts
# gps     - gps coordinates
logtypes=network,csv,xml,weak,cisco,gps

# Do we track probe responses and merge probe networks into their owners?
# This isn't always desireable, depending on the type of monitoring you're
# trying to do.
trackprobenets=true

# Do we log "noise" packets that we can't decipher?  I tend to not, since
# they don't have anything interesting at all in them.
noiselog=false

# Do we log corrupt packets?  Corrupt packets have enough header information
# to see what they are, but someting is wrong with them that prevents us from
# completely dissecting them.  Logging these is usually not a bad idea.
corruptlog=true

# Do we log beacon packets or do we filter them out of the dumpfile
beaconlog=true

# Do we log PHY layer packets or do we filter them out of the dumpfile
phylog=true

# Do we mangle packets if we can decrypt them or if they're fuzzy-detected
mangledatalog=true

# Do we do "fuzzy" crypt detection?  (byte-based detection instead of 802.11
```

```
# frame headers)
# valid option: Comma seperated list of card types to perform fuzzy detection
#  on, or 'all'
fuzzycrypt=wtapfile,wlanng,wlanng_legacy,wlanng_avs,hostap,wlanng_wext,ipw2200,ip-
w2915

# Do we do forgiving fuzzy packet decoding?  This lets us handle borked drivers
# which don't indicate they're including FCS, and then do.
fuzzydecode=wtapfile,radiotap_bsd_a,radiotap_bsd_g,radiotap_bsd_bg,radiotap_bsd_b,p
capfile

# Do we use network-classifier fuzzy-crypt detection?  This means we expect
# packets that are associated with an encrypted network to be encrypted too,
# and we process them by the same fuzzy compare.
# This essentially replaces the fuzzycrypt per-source option.
netfuzzycrypt=true

# What type of dump do we generate?
# valid option: "wiretap"
dumptype=wiretap
# Do we limit the size of dump logs?  Sometimes ethereal can't handle big ones.
# 0 = No limit
# Anything else = Max number of packets to log to a single file before closing
# and opening a new one.
dumplimit=0

# Do we write data packets to a FIFO for an external data-IDS (such as Snort)?
# See the docs before enabling this.
#fifo=/tmp/kismet_dump

# Default log title
logdefault=Kismet

# logtemplate - Filename logging template.
# This is, at first glance, really nasty and ugly, but you'll hardly ever
# have to touch it so don't complain too much.
#
# %n is replaced by the logging instance name
# %d is replaced by the current date as Mon-DD-YYYY
# %D is replaced by the current date as YYYYMMDD
# %t is replaced by the starting log time
# %i is replaced by the increment log in the case of multiple logs
# %l is replaced by the log type (dump, status, crypt, etc)
# %h is replaced by the home directory
# ie, "netlogs/%n-%d-%i.dump" called with a logging name of "Pok" could expand
# to something like "netlogs/Pok-Dec-20-01-1.dump" for the first instance and
# "netlogs/Pok-Dec-20-01-2.%l" for the second logfile generated.
# %h/netlots/%n-%d-%i.dump could expand to
# /home/foo/netlogs/Pok-Dec-20-01-2.dump
#
# Other possibilities:  Sorting by directory
```

```
# logtemplate=%l/%n-%d-%i
# Would expand to, for example,
# dump/Pok-Dec-20-01-1
# crypt/Pok-Dec-20-01-1
# and so on.  The "dump", "crypt", etc, dirs must exist before kismet is run
# in this case.
logtemplate=/var/log/kismet/%n-%d-%i.%l

# Where do we store the pid file of the server?
piddir=/var/run/

# Where state info, etc, is stored.  You shouldnt ever need to change this.
# This is a directory.
configdir=/var/log/kismet/

# cloaked SSID file.  You shouldn't ever need to change this.
ssidmap=ssid_map

# Group map file.  You shouldn't ever need to change this.
groupmap=group_map

# IP range map file.  You shouldn't ever need to change this.
ipmap=ip_map
```

Swatch Start-Up: add to /etc/rc.local

```
# Start swatch
echo -n " swatch"
/usr/local/bin/swatch --config-file /etc/swatchrc --tail-file
/var/log/kismet/alerts/current 2>&1 &
```

A Basic Swatch Configuration: /etc/swatch

```
watchfor /Suspicious client ([0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2}:[0-9A-
F]{2}:[0-9A-F]{2})/
    threshold track_by=$1,type=both,count=5,seconds=604800
    mail "nobody@example.com",subject="Suspicious Client"

watchfor /Out-of-sequence BSS timestamp on ([0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2}:[0-
9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2})/
    threshold track_by=$1,type=both,count=1,seconds=300
    mail "nobody@example.com",subject="Potential AP Spoof"

watchfor /Netstumbler.*from ([0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2}:[0-9A-
F]{2}:[0-9A-F]{2})/
    threshold track_by=$1,type=both,count=1,seconds=300
    mail "nobody@example.com",subject="Netstumbler Probe"

watchfor /Deauthentication\/Dissassociate.*on ([0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2}:
```

```
[0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2})/
    threshold track_by=$1,type=both,count=1,seconds=300
    mail "nobody@example.com",subject="Deauth Flood"

watchfor /Beacon on ([0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2}:
[0-9A-F]{2})/
    threshold track_by=$1,type=both,count=1,seconds=300
    mail "nobody@example.com",subject="AP Changed Channel"

watchfor /Broadcast.*on ([0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]
{2}:[0-9A-F]{2})/
    threshold track_by=$1,type=both,count=1,seconds=300
    mail "nobody@example.com",subject="Broadcast Dissassociation"

watchfor /Suspicious traffic on ([0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2}:
[0-9A-F]{2}:[0-9A-F]{2})/
    threshold track_by=$1,type=both,count=1,seconds=300
    mail "nobody@example.com",subject="Potentail DoS Dissassociation"

watchfor /Illegal SSID length.*from ([0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]{2}:[0-9A-F]
{2}:[0-9A-F]{2}:[0-9A-F]{2})/
    threshold track_by=$1,type=both,count=1,seconds=300
    mail "nobody@example.com",subject="Long SSID"
```

Last Updated: May 23rd, 2017

Upcoming Training

SANS Network Security 2017	**Las Vegas, NV**	**Sep 10, 2017 - Sep 17, 2017**	**Live Event**
Pen Test Hackfest Summit & Training 2017	**Bethesda, MD**	**Nov 13, 2017 - Nov 20, 2017**	**Live Event**