Global Information Assurance Certification Paper

Wireless Security and Monitoring for the Home Network

Raymond Turner

Version 1.4b GIAC GSEC Practical Assignment
August 21, 2003

Abstract

Marketing trends estimate that by the end of 2006, 21 million homes will have implemented a Local Area Network (LAN), and of those 21 million homes 65% will use wireless solutions. [1] The rapidly decreasing cost for wireless devices and the proliferation of wireless solutions provided by the major Internet Service Providers seems to clearly support these growth estimates.

Home wireless users and security professionals the world over are conceptually trying to solve similar problems. They both need to find a way to provide a secure working environment. There are two distinct approaches to this security dilemma, security prevention, and security detection. An example of security prevention would be a firewall device that restricts specific traffic or ports to or from specific hosts. Although this provides protection against unauthorized traffic, it has no means for determining if an attack is being attempted via an authorized port. An example of security detection would be an IDS (Intrusion Detection System) device that contains a signature to identify a specific attack via authorized or unauthorized ports. [2] Security professionals often have the technology and resources to develop security solutions based on prevention, detection, or a combination of the two. However, home wireless users do not have the luxury of evaluating their security approach since the guidelines and wireless devices marketed to the home user demographic have an overwhelming dependency on preventative mechanisms. The first part of this document will briefly review the basic home access point security mechanisms, and their weaknesses. The second part will cover the implementation of a script to detect, identify, and provide notification of users on a home wireless network, as an attempt at security detection.

Preventative Wireless Security

802.11b devices seem to have the market share of home wireless solutions. This is most likely contributed to the steady decrease in the cost for these devices. The top selling 802.11b wireless access points offer basic wireless security features such as WEP, SSID Broadcast, and MAC Filtering. [3] We know from research that users are not implementing these features, [4] and even if they were, a quick review of these basic security features and their weaknesses will help us understand the concerns with relying solely on a preventative approach to wireless security.

WEP

Wired networks offer the luxury of physical boundaries, users wishing to join a wired network must find somewhere to plug-in. It is highly unlikely that a random user on the street will walk in your front door, sit down at your desk, and plug into the hub connected to you cable modem. Wireless networks however do not have this physical connectivity restriction. Neighbors and casual passer bys could connect to your wireless access point without much resistance. The IEEE 802.11 standard specifies WEP otherwise known as Wired Equivalent Privacy for data protection on 802.11 networks. [5] This is proposed to operate as the wireless equivalent to the physical security provided by wired networks. The WEP encryption scheme uses shared keys for the encryption and decryption of the frames passed across a Wireless LAN (WLAN). [6] Some basic rules for implementing WEP include:

1. Turn WEP on. It is not enabled by default and will require configuration on both the client and the access point.
2. Users should define the strongest encryption supported by both the client and the access point.
3. If possible users should change the default cryptographic key, and then schedule key changes daily or weekly.[7]

Although WEP is based on the robust RC4 symmetric key algorithm, the flaws in the implementation of WEP have been well documented. [8] These flaws allow a malicious user who collects enough WEP encrypted frames on given network to identify shared values among the frames and ultimately determine the shared key. It is not secret that WEP has its faults but its implementation is a key piece to a defense in depth approach to home wireless security.

SSID Broadcast

SSID or Service Set Identifier is a unique identifier specified in the header of wireless packets to act as a password for client connectivity to a wireless access point. This is commonly referred to as the wireless network name, and is broadcast on the wireless network by the access point.[9] The following are guidelines for configuring SSID Broadcast on an AP.

1. Unlike WEP turn the SSID Broadcast off if possible
2. Change the default SSID name
3. Increase beacon interval to the maximum setting to make passive scanning more difficult. [7]

The above guidelines only provide protection against the casual snooper. Increasing the beacon interval makes for a quieter access point, and increases the time between each SSID transmission, but as noted the access point still transmits the SSID. While changing the default SSID helps mitigate against

accidental associations with your neighbors access point, which more often than not happens to be manufactured by the same vendor. It also prevents users from easily guessing the SSID when SSID broadcast is disabled. However malicious users passively watching communication on a wireless network can still determine a changed SSID since it transmitted in every associate request and response frame. The features used to obfuscate a wireless network SSID does not provide much in the way of security however it is another key piece in a layered approach to wireless security.

MAC Filtering

A MAC (Media Access Control) is the unique hardware address assigned to every network adapter. The MAC address uniquely identifies each host on a wireless network. MAC Filtering is the process of creating an Access Control List (ACL) to specifically permit or deny certain MAC addresses from connecting to the AP. Listed below are a few important tips about MAC Filtering.

1. Enable MAC Filtering, it won't work otherwise.
2. Since home networks are generally limited to a handful of devices it is likely to be more efficient to create a permit ACL for the known devices on your network rather than a deny ACL for unknown devices.

MAC Filtering does not come without its issues. Address Resolution Protocol or ARP is the protocol used to determine the MAC to IP pairing for the hosts on the wireless network. ARP information is passed in the clear between the clients and the AP. It is only a matter time before a malicious user will discover a MAC address that is permitted to connect to the access point. Some of the most common ARP related attacks are sniffing, hijacking, broadcasting, DOS, and cloning. [10] All is not lost, and although MAC Filtering appears to be no better at protecting wireless networks than WEP and SSID Broadcast, it is a key element in the layered approach to wireless security.

Physical Architecture

The physical implementation of a wireless network can sometimes be used as a method of preventative security. Wireless access points are often considered for implementation in an environment in which a wired network already exists. In this scenario the wireless network can be implemented as an extension of a wired network, or as a separate network or DMZ (De-Militarized Zone).

As an extension of a wired network, little work will need to be done to allow wired and wireless hosts to communication with each other. However the security implications associated with a wireless network would provide a risk to a wired network by circumventing the need for physical connectivity. As a separate network or DMZ the segregation of the two networks, and the restriction of

communication between them could be used to maintain the integrity of the wired network.

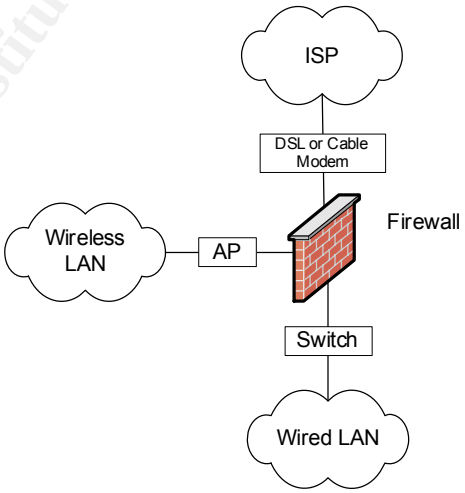
Although it seems futile to implement the above wireless security mechanisms, it is safe say that any security is better than none. While basic wireless security can be compromised, the proliferation of wireless devices and the simple act of implementing WEP, SSID Broadcast, and MAC Filtering, may prompt a malicious user to seek an easier target.

Monitoring or Wireless Security Detection

Most users will never know if someone has accessed their wireless network. The rest of this document will outline an exercise in implementing a simple script to provide a method of monitoring, and reporting wireless network access to the home administrator.

The following assumptions are used in this exercise and should help provide a template for implementing a robust home wireless security solution in which the script will operate.

1. A segregated wireless and wired network will be implemented.
2. All preventative wireless security mechanisms (WEP, SSID Broadcast, MAC Filtering) have been implemented.
3. A firewall provides separation between the two networks.
4. The firewall provides Internet access for each network.
5. Communication between the two networks is prohibited.
6. The firewall will provide addressing for each network.
7. The firewall is used as the monitoring device.



Network topology diagram

Based on the assumptions for this exercise the following tasks are required to support the development of the script:

1. Build a Unix firewall
2. Install the packages used by the script
3. Create and implement the monitoring script

Firewall

For this exercise I have implemented an OpenBSD operating system using pf as the firewall software. The OpenBSD project was chosen for its proactive security stance, and their standard "Only one remote hole in the default install, in more than 7 years!" The instructions for a default install of OpenBSD can be obtained at the OpenBSD's website. [11] Once the default install of OpenBSD is done, the following steps will need to be completed to support the wireless architecture.

1. Disable inetd.conf. This file is the Internet server configuration database and contains the network services like ftp, telnet, and echo. Since SSH provides a more secure method of network connectivity to this box there should not be an instance in which any of the services provided by inetd.conf will be needed. While this is not a necessary step in completing this exercise, it will provide additional security. The following is the easiest way to disable inetd.conf.

    A. Make a copy of the original inetd.conf file and then create a new empty inetd.conf file

```
cd /etc
mv inetd.conf inetd.conf.dist
touch inetd.conf
```

    B. In /etc/rc.conf set inetd to equal NO

```
# set the following to "YES" to turn them on
rwhod=NO
nfs_server=NO           # see sysctl.conf for nfs client configuration
lockd=NO
gated=NO
amd=NO
pf=YES                  # Packet filter / NAT
portmap=NO              # Note: inetd(8) rpc services need portmap too
inetd=NO                # almost always needed
check_quotas=YES        # NO may be desirable in some YP environments
ntpd=YES                # run ntpd if it exists

krb5_master_kdc=NO      # KerberosV master KDC. Run 'info heimdal' for
help.
krb5_slave_kdc=NO       # KerberosV slave KDC.
afs=NO                  # mount and run afs
```

2. Configure the firewall software. Instructions for configuring pf on OpenBSD can be obtained at the OpenBSD website. [12] The pf user's guide also provides a sample rule set.

3. Configure the firewall to be the DHCP server on the wireless network. Both the firewall and most access points can be configured as a DHCP server. Choosing the firewall for the DHCP server will give you more control and future options. For instance perhaps you would like to modify the monitoring script for future interactions with the DHCP leases file. This could allow you to control the issuing of IP leases to trusted hosts only, or to end the lease of a suspect host. The instructions for configuring an OpenBSD DHCP server can be obtained at the OpenBSD website. [13] When configuring the IP ranges in the DHCP server remember that limiting the number of lease addresses can help prevent the casual freeloader from obtaining an IP address on the wireless network.

Packages

All the packages required to create the monitoring script are available in the OpenBSD ports tree.

1. Download the ports tree from OpenBSD and extract them in the /usr directory. Be sure to replace *release#* with the release number of the OpenBSD install.

```
cd /usr
ftp ftp://ftp.openbsd.org/pub/OpenBSD/release#/ports.tar.gz
tar zxvf ports.tar.gz
rm ports.tar.gz
```

2. Install Arpwatch. Arpwatch is a handy utility which monitors MAC to IP pairings on a network. It maintains the pairing information in a .dat file and is configured to report specific changes to syslog and root via email.

```
cd /usr/ports/net/arpwatch
make
make install
```

To make sure that Arpwatch starts on boot append /etc/rc.local with:

```
if [ -x /usr/local/sbin/arpwatch ]; then
            echo -n ' arpwatch';       /usr/local/sbin/arpwatch -i fxp0
fi
```

Replace fxp0 with the firewall interface that will reside in the wireless network.

3. Install Fping. Fping Is a utility similar to ping which uses ICMP echo request to see if a host is up. Fping was chosen because its output is easily parsed, making it a good candidate for use in scripts.

```
cd /usr/ports/net/fping
make
make install
```

4. Install Xprobe. Xprobe is an active OS fingerprinting utility, using ICMP for OS discovery. Xprobe was chosen because it is fast, efficient, and seems to be more consistent at accurately fingerprinting an OS when compared to nmap.

```
cd /usr/ports/net/xprobe
make
make install
```

5. Install Nmap. Nmap is a network mapping and security auditing tool, which uses raw packets to obtain information about hosts on the network. Nmap was chosen to help identify interesting information about non-Windows hosts. If OpenBSD was installed without x11 then make sure you edit the Makefile in the /usr/ports/net/nmap and specify no_x11 for FLAVOR?= before you run make make install.

```
cd /usr/ports/net/nmap
vi Makefile
FLAVORS=        no_x11
FLAVOR?=        no_x11
```

```
cd /usr/ports/net/nmap
make
make install
```

6. Install NBTScan. NBTScan is a utility used to obtain NetBios name information, by sending NetBios status queries. NBTScan was chosen because it identifies interesting information about Windows hosts on the network in simple format.

```
cd /usr/ports/net/nbtscan
make
make install
```

Script

The primary goals of the script are:

1. Provide a way to identify new host on a wireless network
2. Collect information that is useful in identifying hosts. This information might also in determining the validity of the hosts.
3. Provide a method of notification to the home administrator

The script is designed to take the MAC to IP pairing collected by arpwatch and determine if any of those hosts are currently active on the wireless network. The script will then fingerprint the live hosts and use different utilities based on OS to collect interesting information about each host. After this information is collected it will be emailed to a specified recipient. The following pages will first break down the script to show how each part works, followed by a listing of the entire script at the end. Since this script is short and simple, Bourne shell will be used as the scripting language. Future enhancements such as interactions with DHCP leases or pf rule sets may necessitate the need to use a more featured scripting language like perl.

Create the script in the arpwatch directory to keep all output in one place. The script has been broken out into six different sections.

1. The following section will provide the mechanism for collecting the MAC to IP pairing used by the script.

```
#!/bin/sh

# Get MAC to IP list from arp.dat for notify email
cd /var/arpwatch
echo "-----------------------------------------------" >> notify
echo "" >> notify
echo "MAC to IP from arp.dat" >> notify
cut -f 1,2 arp.dat | sort | grep -v 192.168.100.254 >> notify
echo "" >> notify
echo "-----------------------------------------------" >> notify
```

The first line of script specifies the interpreter preceded by #!, A second or empty line was added to set the interpreter apart from the rest of the script.

The third line is commented out and is not executed by the script. Its purpose is to provide a brief explanation of what the next few lines of the script are doing.

The fourth line tells the script to change to the working directory. The working directory in this instance is the directory in which the script is located.

The fifth, sixth, and seventh lines are used to create the *notify* file. This is done using the echo command which in this instance is also provides formatting of the *notify* file.

Arpwatch stores Mac to IP pairing information in a file call *arp.dat.* Here is sample output of an *arp.dat* file. The first two fields are MAC and IP, and the last two are date and name resolution.

```
0:1:2:cc:5a:3   192.168.100.254   1061007749      firewall-int02
0:40:5:c1:5c:8  192.168.100.54    1061007726
0:30:65:1:ca:55 192.168.100.125   1059337083
0:30:65:1:ca:55 192.168.100.53    1059338280
```

```
0:40:5:c1:a8:5  192.168.100.50    1060912812
```

The eighth line uses the cut command to collect the MAC and IP pairing information in the *arp.dat* file. In this instance cut is used with the –f option, which will look for fields separated by tabs. The cut command also uses the 1,2 option to select only the 1st and 2nd field. The grep command is used with the –v option to remove the firewall interface from the list. This information is then appended to the *notify* file.

The ninth and tenth lines provide additional formatting of the *notify* file to signal the end of the collected *arp.dat* information.

2. This section of the script will take the IPs from the *arp.dat* file and determine which hosts are active on the network.

```
# Find live hosts in arp.dat
UPHOSTS=`cut -f 2 /var/arpwatch/arp.dat| grep -v 192.168.100.254`
for LINE in $UPHOSTS; do
/usr/local/sbin/fping -r 1 $LINE | grep alive | awk '{print $1}' >>
up.hosts
done
```

The first line again is commented out and will not be executed by the script. It gives a brief description of what the next few lines of the script will do.

The second line uses the cut command to select the second field in the arp.dat file to create the variable $UPHOSTS. This field in the *arp.dat* file contains the IP portion of the MAC to IP pairings. The grep command with the –v option is used to remove the firewall interface.

Fping is the core utility in next few lines of the script. It is used to determine if a host is alive. Here is sample output of fping.

```
# fping 192.168.100.50
192.168.100.50 is alive
```

The third, fourth, fifth, and sixth lines use a *for* command to tell the script to do the following to each IP in the $UPHOSTS variable.

A. First, fping the IP. The –r option is used so that fping will only attempt to retry pinging once. This is to minimize the amount of time it takes for an IP that does not respond to timeout. Depending on the reliability of the network being monitored, the value specified for the –r can be changed to make sure all alive hosts are identified.
B. Next, the grep command is used to select only the hosts that are alive from the fping.

C. Last, the awk command with the print $1 option will append the first part of the output to the *up.hosts* file. Looking at the sample above the first part of the output would be the IP.

3. Xprobe, Nmap, and NBTScan are the core utilities for this section of the script. Here is a sample of the default output of each.

Xprobe

```
# xprobe 192.168.100.50
X probe ver. 0.0.2
------------------
Interface: xl0/192.168.100.254

LOG: Target: 192.168.100.50
LOG: Netmask: 255.255.255.255
LOG: probing: 192.168.100.50
LOG: [send]-> UDP to 192.168.100.50:32132
LOG: [98 bytes] sent, waiting for response.
LOG: [send]-> ICMP echo request to 192.168.100.50
LOG: [68 bytes] sent, waiting for response.
FINAL:[ Novell (FreeBSD 4.3-current(?) ]
```

Nmap

```
tidalgate# nmap 192.168.100.50

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on  (192.168.100.50):
(The 1600 ports scanned but not shown below are in state: closed)
Port       State       Service
80/tcp     open        http

Nmap run completed -- 1 IP address (1 host up) scanned in 10 seconds
```

NBTScan

```
# nbtscan 192.168.100.54
Doing NBT name scan for adresses from 192.168.100.54

IP address       NetBIOS Name    Server    User      MAC address
------------------------------------------------------------------------
192.168.100.54   JOEPC          <server>  JDOE      0:40:5:c1:5c:8
```

The above utilities have been implemented in the script section below to probe the IPs in the *up.hosts* file, and then based on the OS, report the interesting information about the host.

```
# Probe hosts that are up
HOSTIPS=`cat up.hosts`
for LINE in $HOSTIPS; do
echo "" >> notify
/usr/local/bin/xprobe -o probe.tmp $LINE >> /dev/null 2>&1
cat probe.tmp | grep Target >> notify
cat probe.tmp | grep FINAL >> notify
PROBE=`cat probe.tmp | grep Windows`
```

```
  if [ "$PROBE" = "" ]; then
     /usr/local/bin/nmap -n $LINE | grep -v -i nmap >> notify
  else
     /usr/local/bin/nbtscan -q $LINE >> notify
     echo "" >> notify
  fi
rm probe.tmp
echo "-----------------------------------------------" >> notify
done
```

The first line is commented out and will not be executed by the script. It describes what the following lines of the script will be doing.

The second line uses the cat command to read all the entries in the *up.hosts* file to create the $HOSTIPS variable.

The third through the seventeenth lines use another *for* command to tell the scripts to do the following to each IP in the $HOSTIPS variable.

A. Xprobe the IP to determine the OS. Xprobe uses the –o option to specify *probe.tmp* as a temporary logfile in which the probe results will be written.
B. Use the grep command to gather only the Target information that lists the probed IP and the FINAL information that lists the OS of the probed IP from the *probe.tmp* file. Then append this information to the *notify* file.
C. Check the probed IP to see if it is a windows box. The if statement tells the script that if the probed IP is not a windows then run nmap against the IP and append the results to the *notify* file. If the probed IP is a windows box then the script will run nbtscan against the IP and append the results to the *notify* file.
D. Clean up the *probe.tmp* file for the next entry in the HOSTIPS variable and provide some formatting to the *notify* file with echo to signal the end of the information about the probed IP.

4. The next section of the script will check for the *notify.last* file and will create it if not found.

```
# Check to see if notify.last exist
if [ ! -e "notify.last" ]
then
        touch "notify.last"
fi
```

The first line is a commented out and will not be executed by the script. It describes what the following lines of the script will be doing.

The second, third, fourth, and fifth lines use an *if* statement to tell the script to see if the *notify.last* file exists. If the *notify.last* file does not then the script will create the file. The *notify.last* file contains the information collected during the last time the script was run.

5. The next section is responsible for determining if there have been any changes on the wireless network.

```
# Compare notify with notify.last if nothing to compate then quite
COMP=`diff -q notify notify.last`
if [ $COMP = "" ]; then
   rm notify
   rm up.hosts
   exit 0
else
   cat notify | mail -s "ARP Traffic" root
   cat notify > notify.last
fi
```

The first line is a commented out and will not be executed by the script. It describes what the following lines of the script will be doing.

The second line uses the diff command to determine if there are differences between the *notify* file and the *notify.last* file. The –q option is used so that diff only reports if the files differ, not the details of the differences. The output is used to create the $COMP variable.

The third through the tenth lines use an *if* command to tell the script to do the following.

A. If there is no difference between *notify* and *notify.last* then remove the *notify* and *up.hosts* file and exit the script.
B. If there is a difference between *notify* and *notify.last* then use the cat command to read the *notify* file and pipe the results into an email sent to a specified recipient (i.e. home administrator) which happens to be root in the above instance. The script will then use the cat command to overwrite the *notify.last* file with the information in the *notify* file.

6. This section of the script is performing some remaining clean up.

```
# Clean up
rm up.hosts
rm notify
```

The first line is a commented out and will not be executed by the script. It describes what the following lines of the script will be doing.

The second and third lines remove the *up.hosts* and *notify* files so they can be created from scratch the next time the script runs.

The following page shows the script in its entirety.

```
#!/bin/sh

# Get MAC to IP list from arp.dat for notify email
cd /var/arpwatch
echo "-------------------------------------------------" >> notify
echo "" >> notify
echo "MAC to IP from arp.dat" >> notify
cut -f 1,2 arp.dat | sort | grep -v 192.168.100.254 >> notify
echo "" >> notify
echo "-------------------------------------------------" >> notify

# Find live hosts in arp.dat
UPHOSTS=`cut -f 2 /var/arpwatch/arp.dat| grep -v 192.168.100.254`
for LINE in $UPHOSTS; do
/usr/local/sbin/fping -r 1 $LINE | grep alive | awk '{print $1}' >>
up.hosts
done

# Probe hosts that are up
HOSTIPS=`cat up.hosts`
for LINE in $HOSTIPS; do
echo "" >> notify
/usr/local/bin/xprobe -o probe.tmp $LINE >> /dev/null 2>&1
cat probe.tmp | grep Target >> notify
cat probe.tmp | grep FINAL >> notify
PROBE=`cat probe.tmp | grep Windows`
  if [ "$PROBE" = "" ]; then
     /usr/local/bin/nmap -n $LINE | grep -v -i nmap >> notify
  else
     /usr/local/bin/nbtscan -q $LINE >> notify
     echo "" >> notify
  fi
rm probe.tmp
echo "-------------------------------------------------" >> notify
done

# Check to see if notify.last exist
if [ ! -e "notify.last" ]
then
        touch "notify.last"
fi

# Compare notify with notify.last if nothing to compate then quite
COMP=`diff -q notify notify.last`
if [ $COMP = "" ]; then
   rm notify
   rm up.hosts
   exit 0
else
   cat notify | mail -s "ARP Traffic" root
   cat notify > notify.last
fi

# Clean up
rm up.hosts
rm notify
```

Once the script has been completed, the last step is to create a cron job so that it runs on regularly scheduled interval. The frequency at which the script should be run is a matter of preference. I found that a five minute interval keeps excess network traffic to minimum but still provides a reasonable window for detecting unwanted hosts.

Below is a sample of a notification email. The first part of the email includes the MAC to IP pairing from the *arp.dat* file. This information allows you to review the history of access to the wireless network. The second and third parts are an NBTScan of a windows host, and an Nmap port scan of non-Windows host on the wireless network. This is used to provide information about each host and has the potential to be used for host validation. It is important to understand that host validation is dependant upon an understanding of the wireless network and the devices that will have access. Obviously if you don't know that an authorized user uses a Windows XP laptop named JOEPC with the username JDOE then the notification email will not be useful in determining if a malicious user possibly spoofed a valid MAC address.

```
-----------------------------------------------

MAC to IP from arp.dat
0:1:2:cc:5a:3   192.168.100.254
0:40:5:c1:5c:8  192.168.100.54
0:30:65:1:ca:55 192.168.100.125
0:30:65:1:ca:55 192.168.100.53
0:40:5:c1:a8:5  192.168.100.50

-----------------------------------------------

LOG: Target: 192.168.100.54
FINAL:[ Windows 2k. SP1, SP2/Windows XP ]
192.168.100.54   JOEPC        <server>    JDOE    00-40-05-c1-5c-8

-----------------------------------------------

LOG: Target: 192.168.100.50
FINAL:[ Novell (FreeBSD 4.3-current(?) ]

Interesting ports on  (192.168.100.50):
(The 1600 ports scanned but not shown below are in state: closed)
Port       State       Service
80/tcp     open        http

-----------------------------------------------
```

Conclusion

This practicum reviewed security prevention and security detection options for home wireless users. Although the monitoring script offered in this document could be used to monitor home wired networks, the information carries more

weight in a wireless network due to the anonymity of wireless connections. [14] Similar to the security issues with WEP, SSID Broadcast, and MAC Filtering, the methodology used by the monitoring script for host validation is not without its flaws. The script accomplishes the goal of providing a simple and cost effective method of detection, identification, and notification of wireless hosts. The means for developing a more robust method for host validation remains opened for future revisions.

References

[1] Scherf, Kurt. “Trends and Outlook for Wireless Home Networks.” Parks Associates. Aug 2002.
URL: http://www.parksassociates.com/inthePress/resources/resources.htm

[15] Deshon, Markus. “Intrusion Prevention versus Intrusion Detection” SecureWorks.
URL:http://www.secureworks.net/techResourceCenter/fullTechArticle.php?article=IpsVsIds

[7] Mitchell, Bradley “Top 6 802.11b Wireless Access Points for Home.” About.com
URL:http://compnetworking.about.com/cs/wireless80211/tp/80211b_aps_home.htm

[2] Fisher, Ken. “Security Practicum: Home Wireless Security Practices.” Ars Technica.
URL: http://www.arstechnica.com/paedia/w/wireless-security-howto/home-802.11b-1.html

[8] IEEE “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.” ANSI/IEEE 1999
URL:http://www.techfree.com/802%20standards/802.11%20Wireless%20LAN/802.11 -1999.pdf

[10] Geier, Jim. “802.11 WEPL Concepts and Vulnerability” 802.11-Planet.com. 20 June, 2002
URL: http://www.80211-planet.com/tutorials/article.php/1368661

[4] Leres, Craig. “Arpwatch” Lawrence Berkeley National Laboratory Network Research Group. Oct 2000
URL: ftp://ftp.ee.lbl.gov/arpwatch.tar.gz

[9] Fluhrer, Scott, Mantin, Itsik and Shamir, Adi “Weaknesses in the Key Scheduling Algorithm of RC4” Lecture Notes in Computer Science. 2001
URL: http://downloads.securityfocus.com/library/rc4_ksaproc.pdf

[6] “SSID” Webopedia
URL: http://www.webopedia.com/TERM/S/SSID.html

[11] Whalen, Sean. “An Introduction to ARP Spoofing” Apr 2001
URL: http://www.node99.org/projects/arpspoof/arpspoof.pdf

[12] “4 – OpenBSD 3.3 Installation Guide” OpenBSD.org. Jul 9, 2003
URL: http://www.openbsd.org/faq/faq4.html

[13] “PF: The OpenBSD Packet Filter” OpenBSD.org. Jun 29, 2003.
URL: http://www.openbsd.org/faq/pf/index.html

[14] “6.4.2 DHCP Server” OpenBSD.org. Jul 25, 2003.
URL: http://www.openbsd.org/faq/faq6.html#DHCP

[5] WECA “The Wireless Ethernet Compatibility Alliance” WLANA. Sept 7, 2001.
URL: http://www.wlana.org/learn/security.htm