Detecting Wireless LAN MAC Address Spoofing

Joshua Wright, GCIH, CCNA
Joshua.Wright@jwu.edu
http://home.jwu.edu/jwright/
1/21/2003

Abstract

An attacker wishing to disrupt a wireless network has a wide arsenal available to them. Many of these tools rely on using a faked MAC address, masquerading as an authorized wireless access point or as an authorized client. Using these tools, an attacker can launch denial of service attacks, bypass access control mechanisms, or falsely advertise services to wireless clients.

This presents unique opportunities for attacks against wireless networks that are difficult to detect, since the attacker can present himself as an authorized client by using an altered MAC address. As nearly all wireless NICs permit changing their MAC address to an arbitrary value – through vendor-supplied drivers, open-source drivers or various application programming frameworks – it is trivial for an attacker to wreak havoc on a target wireless LAN.

This paper describes some of the techniques attackers utilize to disrupt wireless networks through MAC address spoofing, demonstrated with captured traffic that was generated by the AirJack, FakeAP and Wellenreiter tools. Through the analysis of these traces, the author identifies techniques that can be employed to detect applications that are using spoofed MAC addresses. With this information, wireless equipment manufacturers could implement anomaly-based intrusion detection systems capable of identifying MAC address spoofing to alert administrators of attacks against their networks.

Introduction

MAC addresses have long been used as the singularly unique layer 2 network identifier in LANs. Through controlled, organizationally unique identifiers (OUI) allocated to hardware manufacturers, MAC addresses are globally unique for all LAN-based devices in use today. In many cases, the MAC address of a workstation is used as an authentication factor or as a unique identifier for granting varying levels of network or system privilege to a user.

This method of client tracking and authentication is also employed in 802.11 wireless networks. Attackers targeting wireless LANs utilize the ability to change their MAC address to circumvent network security measures: an attacker with minimal skill might alter their MAC address in an effort to masquerade or hide their presence, an attacker with minimally more skill might change their MAC address to one that is otherwise authorized to bypass access control lists or to escalate network privileges.

In this paper, I demonstrate two methods of detecting wireless LAN (WLAN) MAC address spoofing. I also show how these methods can be used to detect the activity of devious WLAN attack tools.

Changing MAC Addresses

The phrase "MAC address spoofing" in this context relates to an attacker altering the manufacturer-assigned MAC address to any other value. This is conceptually different than traditional IP address spoofing where an attacker sends data from an arbitrary source address and does not expect to see a response to their actual source IP address. MAC address spoofing might be more accurately described as MAC address "impersonating" or "masquerading" since the attacker is not crafting data with a different source than is their transmitting address. When an attacker changes their MAC address they continue to utilize the wireless card for its intended layer 2 transport purpose, transmitting and receiving from the same source MAC.

Nearly all 802.11 cards in use permit their MAC addresses to be altered, often with full support and drivers from the manufacturer. Using Linux open-source drivers, a user can change their MAC address with the ifconfig tool, or with a short C program calling the ioctl() function with the SIOCSIFHWADDR flag. Windows users are commonly permitted to change their MAC address by selecting the properties of their network card drivers in the network control panel applet.

An attacker may choose to alter their MAC address for several reasons, including obfuscating their presence on a network, to bypass access control lists, or to impersonate an already-authenticated user. Each are explained in greater detail as follows:

Obfuscating network presence: An attacker might choose to change their MAC address in an attempt to evade network intrusion detection systems (NIDS). A common example is an attacker executing a brute-force attack script with a random MAC address for each successive connection attempt. Such an attack would go undetected by network activity analysis applications such as NetFlow that report upper-layer network activity or large quantities of traffic from a single source address.

Bypassing access control lists: Used as a basic form of access control on WLANs, administrators typically have the option to configure access points or neighboring routers to permit only registered MAC addresses to communicate on the network. An attacker could circumvent this form of access control by passively monitoring the network and generate a list of MAC addresses that are authorized to communicate. With the list of authorized MAC addresses in hand, an attacker is free to set their MAC address to any of the authorized addresses, bypassing the intended security mechanism.

Authenticated user impersonation: Certain hardware WLAN security authentication devices rely on matching user authentication credentials to the source MAC address of a client. After a user has successfully authenticated, the security gateway permits traffic based on a dynamic list of authorized MAC addresses. An attacker wishing to circumvent the security of the device only

needs to monitor network activity for an authorized client MAC address and then alter their MAC address to match the authenticated client before communicating on the network.

Detecting Anomalous MAC Addresses

A hardware manufacturer wishing to produce network cards needs to obtain a three-byte organizationally unique identifier from the Institute of Electrical and Electronics Engineers to be used as a prefix for the MAC addresses of their products. This identify string permits a manufacturer to maintain their own allocation procedure for MAC addresses, ensuring they are globally unique. At this time, the IEEE has allocated 6,278 prefixes to various organizations (IEEE, 2002).

The IEEE makes the list of prefix allocations and the assigned company information available to the public, largely for users to match a piece of equipment with a MAC address to its manufacturer. We can use this list to evaluate all source MAC addresses on the network to determine if the prefix is allocated by the IEEE. MAC addresses that appear on the network using a prefix as yet unallocated by the IEEE can be flagged as anomalous activity.

Identifying Wellenreiter

When trying to obfuscate their presence on a network, an attacker will typically generate and utilize a random MAC address before sending traffic. This is demonstrated in Wellenreiter, a wireless LAN discovery application that uses a procedure to brute-force the SSID of an access point. The following Perl extract is taken from Wellenreiter v1.6:

```
# comments by Joshua Wright
sub build_a_fakemac
{
        my $fakemac;
        # Perform 4 iterations of the following statements.  This is actually a bug, should
        # be 5 iterations to generate a 40-bit value.  This procedure will consistently
        # generate MAC addresses that ifconfig will pad with a trailing hex 40.
        for (my $i =0;$i < 4;$i++)
                {
                        # $temp contains a random hex value between 0 and 255
                        my $temp = sprintf "%x", int(rand(255));
                        if (length($temp) == 1)
                        {
                                # prepend single-digit values with a leading zero
                                $temp = '0' . $temp;
                        }
                                # append the hex value in $temp to the generated MAC address
                                $fakemac = $fakemac . $temp;
                }
        # prepend a leading 00 to the generated MAC address to avoid conflict with reserved or
        # multicast/broadcast MAC addresses
        $fakemac = '00' . $fakemac;
        return ($fakemac);
}
```

In this example, Wellenreiter generates 4 random values between “0x00” and “0xFF” and then prepends a leading value of “0x00” to avoid generating MAC addresses that conflict with reserved and multicast space. After generating a random MAC address, Wellenreiter uses the Linux “ifconfig” tool to set the MAC address before trying to associate with the target access

point. A trace of Wellenreiter v1.6 traffic attempting to brute-force the ESSID of an access point appears as follows:

```
tecra:~ $ tethereal –r wellenreiter.dmp –n –R “wlan.fc eq 0x0040”
167  16.117154 00:03:b7:47:e2:40 -> ff:ff:ff:ff:ff:ff IEEE 802.11 Probe Request
168  16.133680 00:03:b7:47:e2:40 -> ff:ff:ff:ff:ff:ff IEEE 802.11 Probe Request
171  16.239248 00:84:b9:5a:ce:40 -> ff:ff:ff:ff:ff:ff IEEE 802.11 Probe Request
177  16.504359 00:84:b9:5a:ce:40 -> ff:ff:ff:ff:ff:ff IEEE 802.11 Probe Request
178  16.538390 00:0e:ef:e0:3d:40 -> ff:ff:ff:ff:ff:ff IEEE 802.11 Probe Request
183  16.885264 00:0e:ef:e0:3d:40 -> ff:ff:ff:ff:ff:ff IEEE 802.11 Probe Request
185  16.918787 00:a6:8b:cd:f1:40 -> ff:ff:ff:ff:ff:ff IEEE 802.11 Probe Request
191  17.270278 00:a6:8b:cd:f1:40 -> ff:ff:ff:ff:ff:ff IEEE 802.11 Probe Request
192  17.303924 00:8a:7c:29:56:40 -> ff:ff:ff:ff:ff:ff IEEE 802.11 Probe Request
198  17.653241 00:8a:7c:29:56:40 -> ff:ff:ff:ff:ff:ff IEEE 802.11 Probe Request
199  17.674149 00:df:f7:a1:8a:40 -> ff:ff:ff:ff:ff:ff IEEE 802.11 Probe Request
206  18.041326 00:df:f7:a1:8a:40 -> ff:ff:ff:ff:ff:ff IEEE 802.11 Probe Request
tecra:~ $
```

In this example every second probe request frame appears to be coming from a different MAC address, all destined to the broadcast address. If we compare the MAC address prefixes to the allocated OUI listing by the IEEE we discover that only one of the five MAC prefixes is listed as being allocated by the IEEE.

Armed with this information, we can detect anomalous MAC addresses based on their prefix information. With a current list of allocated OUIs from the IEEE, we can monitor network activity and identify the OUIs for MAC addresses that are not presently allocated. To demonstrate this detection method, I have written a short Perl program called maidwts (MAC Addresses I Don’t Want To See, Appendix A). This tool uses the Net::Pcap CPAN module to capture raw 802.11 frames in RFMON mode and reports anomalous OUIs. The following output was captured using maidwts while attempting to brute-force the ESSID of an access point using Wellenreiter v1.6:

```
tecra:~ $ sudo iwpriv eth0 monitor 2 11
tecra:~ $ sudo perl maidwts.pl -a -i eth0 -f oui.txt
Reading list of OUI allocations from oui.txt .. Done.
Generating hash .. Done.
Capturing traffic ..
Alert: Unallocated OUI prefix for address 001c978ffe40
Alert: Unallocated OUI prefix for address 001c978ffe40
Alert: Unallocated OUI prefix for address 00c510269c40
Alert: Unallocated OUI prefix for address 00c510269c40
Alert: Unallocated OUI prefix for address 00c510269c40
Alert: Unallocated OUI prefix for address 0092cefcc740
Alert: Unallocated OUI prefix for address 0092cefcc740
Alert: Unallocated OUI prefix for address 0092cefcc740
Alert: Unallocated OUI prefix for address 0083f1701740
Alert: Unallocated OUI prefix for address 0083f1701740
Alert: Unallocated OUI prefix for address 0083f1701740
Alert: Unallocated OUI prefix for address 0068a59cc940
Alert: Unallocated OUI prefix for address 0068a59cc940
Alert: Unallocated OUI prefix for address 0068a59cc940
Alert: Unallocated OUI prefix for address 0036e7ca1640
Alert: Unallocated OUI prefix for address 0036e7ca1640
tecra:~ $
```

We can see that Wellenreiter generated numerous MAC prefixes that do not match the allocation list published by the IEEE. By monitoring this activity and comparing the OUIs to the allocation list, we can detect users who are using randomized MAC prefixes.

Unfortunately, a crafty attacker can easily circumvent this detection technique. Since the list of allocated OUIs is publicly available, an attacker could modify their randomization procedure to build a hash of allocated prefixes and randomly select an allocated OUI, then append a random 24-bit value for the remaining three octets of the MAC address. This technique was first demonstrated in the FakeAP tool that generates multiple SSID beacon frames with varying MAC addresses, power output, and SSID values in an effort to fool network discovery applications into thinking they have discovered a plethora of wireless networks. Fortunately, we can detect the presence of FakeAP and other, more devious WLAN attack tools through WLAN sequence number analysis.

WLAN Sequence Number Analysis

The MAC layer for 802.11 networks is significantly more complex than previous IEEE 802 designations. Design factors including the ability to ensure reliable transport in the presence of interference from an outside source and the ability to transparently roam between basic service sets (BSS) were goals of the IEEE committee that required new fields in the 802.11 frame headers. As part of the 802.11 specifications, the IEEE included a method to accommodate fragmentation for large management and data frames through the use of frame sequencing.

The IEEE dedicated two bytes of every 802.11 frame for sequence control fields: four bits for a fragment number and 12 bits for a sequence number. When management or data frames need to be fragmented they are transmitted in portions with a constant sequence number and incrementing fragment numbers for each portion of the packet. The receiving station reassembles all traffic with a constant sequence number in the sequence control field.

The sequence number field is a sequential counter that is incremented by one for each non-fragmented frame, starting at 0, modulo 4096. The fragment number is always zero unless the frame is a fragment of a larger packet. Figure 1 details the generic 802.11 frame header with an expanded sequence control field.

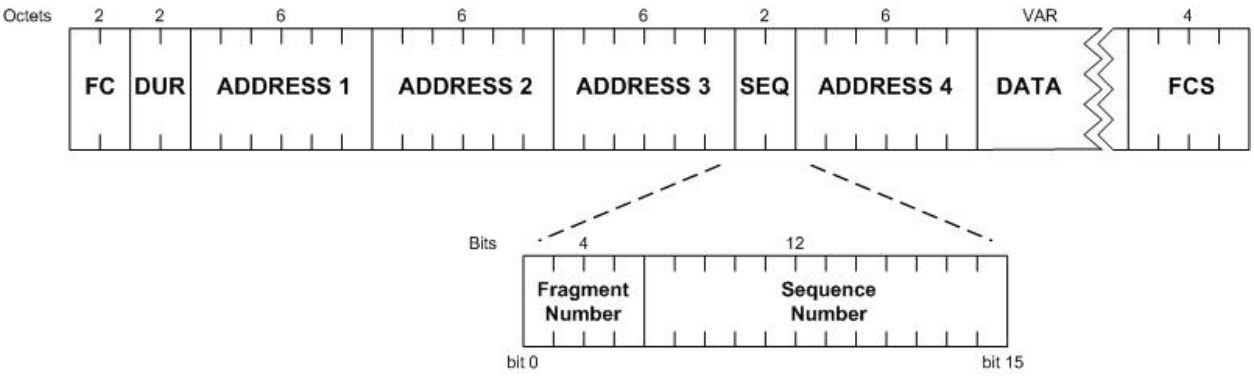

Figure 1

The sequence control field is similar to its upper-layer counterpart, the IP identification field. Unlike the IP identification field however, an attacker crafting custom 802.11 frames does not have the ability to set this parameter to an arbitrary value.

Much of the activity in 802.11 attack tools has been focused around the Prism WLAN HostAP drivers (Malinen), with some preliminary work on a generic wireless packet injection framework by Mike Schiffman called LibRadiate (Schiffman). Both HostAP and LibRadiate utilize the features of Prism-based wireless cards to transmit management and control frames onto the wireless LAN, relying on proprietary card firmware to handle time-sensitive tasks like power management, beaconing, frame acknowledgement, and controlling sequence numbers in transmitted frames (Malinen).

Without the ability to control the firmware functionality of wireless cards, and without the source code to develop custom firmware, an attacker does not have the ability to alter the value of the sequence control field in the 802.11 header. By analyzing sequence number patterns, we can identify traffic that is using a spoofed MAC address without relying on the MAC OUI information. Since this is best explained through example, I will demonstrate how to use sequence number analysis to identify traffic generated by the FakeAP and AirJack tools, as well as how to detect MAC address spoofing intended to bypass MAC-based authentication systems.

Identifying FakeAP

FakeAP is a Perl script that utilizes the HostAP client drivers and a dictionary word list to generate 802.11 beacon frames in an attempt to fool NetStumbler, Kismet and other wireless LAN discovery applications (FakeAP, 2002). In order to make the generated traffic appear valid, FakeAP supports changing transmit signal strength and MAC addresses for each unique SSID advertisement. The authors of FakeAP were resourceful in their MAC address generation code, using a list of allocated OUIs with three trailing random octets to generate a MAC address that will escape the anomalous MAC prefix detection method described above.

We can detect FakeAP activity by monitoring the sequence number value for sequential increments with changing source MAC address, BSSID and SSID values. In an environment with a dense deployment of wireless access points, we would typically see multiple beacon frames from varying source addresses containing different SSID names; using FakeAP, we will see similar activity as it cycles through its dictionary list of SSID values and random MAC address generation procedure. The firmware of the Prism wireless NIC will be indifferent to changing wireless parameters, however, maintaining a sequential order to the sequence number value.

The following packet capture was generated in a lab environment using FakeAP v0.3.1, with the output from tethereal trimmed for brevity:

```
tecra:~ $ tethereal –r fakeap.dmp –n –R “wlan.fc eq 0x0080”
IEEE 802.11
    Type/Subtype: Beacon frame (8)
    Destination address: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)
    Source address: 00:02:b3:cb:28:64 (00:02:b3:cb:28:64)
    BSS Id: 00:02:b3:cb:28:64 (00:02:b3:cb:28:64)
    Fragment number: 0
    Sequence number: 2055
IEEE 802.11 wireless LAN management frame
    Tagged parameters (22 bytes)
        Tag Number: 0 (SSID parameter set)
        Tag interpretation: macro
```



```
IEEE 802.11
    Type/Subtype: Beacon frame (8)
    Destination address: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)
    Source address: 00:02:a5:a5:75:a5 (00:02:a5:a5:75:a5)
    BSS Id: 00:02:a5:a5:75:a5 (00:02:a5:a5:75:a5)
    Fragment number: 0
    Sequence number: 2056
IEEE 802.11 wireless LAN management frame
    Tagged parameters (26 bytes)
        Tag Number: 0 (SSID parameter set)
        Tag interpretation: breathing

IEEE 802.11
    Type/Subtype: Beacon frame (8)
    Destination address: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)
    Source address: 00:02:a5:a5:75:a5 (00:02:a5:a5:75:a5)
    BSS Id: 00:02:a5:a5:75:a5 (00:02:a5:a5:75:a5)
    Fragment number: 0
    Sequence number: 2057
IEEE 802.11 wireless LAN management frame
    Tagged parameters (26 bytes)
        Tag Number: 0 (SSID parameter set)
        Tag interpretation: breathing

IEEE 802.11
    Type/Subtype: Beacon frame (8)
    Destination address: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)
    Source address: 00:02:2d:7d:c5:5f (00:02:2d:7d:c5:5f)
    BSS Id: 00:02:2d:7d:c5:5f (00:02:2d:7d:c5:5f)
    Fragment number: 0
    Sequence number: 2058
IEEE 802.11 wireless LAN management frame
    Tagged parameters (29 bytes)
        Tag Number: 0 (SSID parameter set)
        Tag interpretation: intercession
tecra:~ $
```

In this example we see unique source MAC addresses, BSSIDs and SSIDs that would normally appear as traffic generated by multiple WLAN access points. If we examine the sequence number parameter, however, we see a sequential pattern that indicates that this traffic is being generated by a single host and not multiple access points.

While it may be useful to detect the presence of tools such as FakeAP, we can also use sequence number analysis to identify traffic that uses spoofed MAC addresses to bypass security mechanisms or to perform denial-of-service attacks against a target WLAN.

Identifying AirJack

AirJack is a suite of tools that is designed as a proof-of-concept to establish layer 1 man-in-the-middle attacks against 802.11 networks (AirJack, 2002). Included in this toolset is "wlan-jack", a tool to perform a denial-of-service attack against users on a target wireless network; it works by sending spoofed deauthenticate frames to a broadcast address, purportedly from the network access point's MAC address.

Using the wlan-jack tool is simple; an attacker only has to be in range of the clients on the wireless network to be effective. This tool works because the clients that receive the deauthenticate frames believe they have been sent from the access point they are currently

associated with. Since wlan-jack has to spoof the MAC address of the target access point, we have the opportunity to identify this traffic as anomalous based on sequence number analysis.

In order to detect anomalies in sequence numbers, we need to first establish a pattern of legitimate sequence number activity for each MAC address we wish to monitor. In the following detect, the sequence numbers for the legitimate source MAC "00:e0:63:82:19:c6" are in the range 2966 – 2971. Knowing this pattern, we can easily identify the deauthenticate frames as being illegitimate, although they purport to have originated from the source MAC "00:e0:63:82:19:c6". This packet capture was generated in a lab environment using AirJack v0.6.2-alpha.

```
tecra:~ $ tethereal –r airjack.dmp –n –R "wlan.sa eq 00:e0:63:82:19:c6"
IEEE 802.11
    Type/Subtype: Beacon frame (8)
    Destination address: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)
    Source address: 00:e0:63:82:19:c6 (00:e0:63:82:19:c6)
    BSS Id: 00:e0:63:82:19:c6 (00:e0:63:82:19:c6)
    Fragment number: 0
    Sequence number: 2966

IEEE 802.11
    Type/Subtype: Beacon frame (8)
    Destination address: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)
    Source address: 00:e0:63:82:19:c6 (00:e0:63:82:19:c6)
    BSS Id: 00:e0:63:82:19:c6 (00:e0:63:82:19:c6)
    Fragment number: 0
    Sequence number: 2967

IEEE 802.11
    Type/Subtype: Probe Response (5)
    Destination address: 00:60:1d:f0:91:56 (00:60:1d:f0:91:56)
    Source address: 00:e0:63:82:19:c6 (00:e0:63:82:19:c6)
    BSS Id: 00:e0:63:82:19:c6 (00:e0:63:82:19:c6)
    Fragment number: 0
    Sequence number: 2968

IEEE 802.11
    Type/Subtype: Deauthentication (12)
    Destination address: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)
    Source address: 00:e0:63:82:19:c6 (00:e0:63:82:19:c6)
    BSS Id: 00:e0:63:82:19:c6 (00:e0:63:82:19:c6)
    Fragment number: 0
    Sequence number: 1335
IEEE 802.11 wireless LAN management frame
    Fixed parameters (2 bytes)
        Reason code: Previous authentication no longer valid (0x0002)

IEEE 802.11
    Type/Subtype: Deauthentication (12)
    Destination address: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)
    Source address: 00:e0:63:82:19:c6 (00:e0:63:82:19:c6)
    BSS Id: 00:e0:63:82:19:c6 (00:e0:63:82:19:c6)
    Fragment number: 0
    Sequence number: 1336
IEEE 802.11 wireless LAN management frame
    Fixed parameters (2 bytes)
        Reason code: Previous authentication no longer valid (0x0002)

IEEE 802.11
    Type/Subtype: Beacon frame (8)
    Destination address: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)
    Source address: 00:e0:63:82:19:c6 (00:e0:63:82:19:c6)
    BSS Id: 00:e0:63:82:19:c6 (00:e0:63:82:19:c6)
    Fragment number: 0
    Sequence number: 2969
```



```
IEEE 802.11
    Type/Subtype: Probe Response (5)
    Destination address: 00:60:1d:f0:91:56 (00:60:1d:f0:91:56)
    Source address: 00:e0:63:82:19:c6 (00:e0:63:82:19:c6)
    BSS Id: 00:e0:63:82:19:c6 (00:e0:63:82:19:c6)
    Fragment number: 0
    Sequence number: 2970

IEEE 802.11
    Type/Subtype: Deauthentication (12)
    Destination address: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)
    Source address: 00:e0:63:82:19:c6 (00:e0:63:82:19:c6)
    BSS Id: 00:e0:63:82:19:c6 (00:e0:63:82:19:c6)
    Fragment number: 0
    Sequence number: 1339
IEEE 802.11 wireless LAN management frame
    Fixed parameters (2 bytes)
        Reason code: Previous authentication no longer valid (0x0002)

IEEE 802.11
    Type/Subtype: Probe Response (5)
    Destination address: 00:60:1d:f0:91:56 (00:60:1d:f0:91:56)
    Source address: 00:e0:63:82:19:c6 (00:e0:63:82:19:c6)
    BSS Id: 00:e0:63:82:19:c6 (00:e0:63:82:19:c6)
    Fragment number: 0
    Sequence number: 2971

tecra:~ $
```

We can quickly identify the frames with sequence numbers in the range of 1335-1339 as illegitimate management traffic.

Identifying Clients Bypassing Access Control Mechanisms

The third example will demonstrate how an attacker might choose to bypass the security restrictions of MAC address access lists or security gateways through MAC address spoofing.

Where access control lists are useful in restricting the wireless network to only registered MAC addresses, it does not accommodate user authentication credentials before granting access to the network. As an alternative, network authentication appliances utilize dynamic access control, permitting access to those users who successfully authenticate to a backend directory service. Once a user is authenticated, the appliance forwards the traffic from the managed (insecure) to the protected (secure) interface based on the client MAC addresses – which, as we have seen, can be spoofed by an attacker.

The following network trace was generate with two clients: a Windows XP client (Vic) that has successfully authenticated to a WLAN security appliance, and a Slackware Linux attacker (Eve) wishing to access resources on the target network. This attack was capture in a lab environment where Eve is using a Lucent 802.11b card with the manufacturer-assigned MAC address "00:02:2d:09:a1:dd" and IP address "10.21.5.209". Vic is also using a Lucent 802.11b card with a manufacturer-assigned MAC address "00:02:2d:38:83:2c" and IP address "10.21.5.188".

1. Eve discovers an open access point while running Kismet, and wants to access google.com. After associating with the network and receiving an IP address from a

DHCP server, Eve discovers the WLAN is protected by a WLAN security appliance after opening her web browser and being redirected to a page requesting user authentication.

2. With growing curiosity about the target network, Eve starts capturing traffic on her wireless network card with tcpdump:

```
eve:~ $ tcpdump –i eth0 –e –n –X 'ip'
tcpdump: listening on eth0
14:20:41.690462 0:2:2d:38:83:2c 0:3:47:df:12:72 0800 510: 10.21.5.188.1118 >
207.46.200.145.80: P 0:456(456) ack 1 win 15466 (DF)
0x0000   4500 01f0 06e2 4000 8006 4a95 0a15 05bc       E.....@...J.....
0x0010   cf2e c891 045e 0050 442f 4ce0 6e96 9387       .....^.PD/L.n...
0x0020   5018 3c6a 5361 0000 4745 5420 2f67 616d       P.<jSa..GET./gam
0x0030   6573 2f20 4854                                es/.HT
14:20:41.790616 0:3:47:df:12:72 0:2:2d:38:83:2c 0800 60: 207.46.200.145.80 >
10.21.5.188.1118: . ack 456 win 16900 (DF)
0x0000   4500 0028 0556 4000 2e06 9fe9 cf2e c891       E..(.V@.........
0x0010   0a15 05bc 0050 045e 6e96 9387 442f 4ea8       .....P.^n...D/N.
0x0020   5010 4204 2c9c 0000 0000 0000 0000            P.B.,.........
14:20:41.893226 0:3:47:df:12:72 0:2:2d:38:83:2c 0800 409: 207.46.200.145.80 >
10.21.5.188.1118: P 1:356(355) ack 456 win 16900 (DF)
0x0000   4500 018b 05cf 4000 2e06 9e0d cf2e c891       E.....@.........
0x0010   0a15 05bc 0050 045e 6e96 9387 442f 4ea8       .....P.^n...D/N.
0x0020   5018 4204 4002 0000 4854 5450 2f31 2e31       P.B.@...HTTP/1.1
0x0030   2032 3030 204f                                .200.O
14:20:41.896315 0:3:47:df:12:72 0:2:2d:38:83:2c 0800 1354: 207.46.200.145.80 >
10.21.5.188.1118: . 356:1656(1300) ack 456 win 16900 (DF)
0x0000   4500 053c 05d0 4000 2e06 9a5b cf2e c891       E..<..@....[....
0x0010   0a15 05bc 0050 045e 6e96 94ea 442f 4ea8       .....P.^n...D/N.
0x0020   5010 4204 64c1 0000 0d0a 3c21 2d2d 204d       P.B.d.....<!--.M
0x0030   6963 726f 736f                                icroso
```

With this information Eve determines that Vic is active on the wireless network and records his IP address, MAC address and default gateway information. From passive-fingerprint information and the User-Agent information available from web-browser activity, Eve surmises that Vic is a Windows XP workstation.

3. Eve launches a well-published DoS attack against Vic, causing his terminal to crash with the blue screen of death. Eve then immediately changes her MAC address, IP address and default gateway to the values Vic was using.

```
eve:~ $ smbnuke 10.21.5.188 –t 1 –T 2
Windows SMB Nuker (DoS) - Proof of concept - CVE CAN-2002-0724
Copyright 2002 - Frederic Deletang (df@phear.org) - 28/08/2002

Trying to list netbios names on 10.21.5.188
Using netbios name: 5Y28KSQZG0D3
Connecting to remote host (10.21.5.188:139)...
Negotiating protocol...
Requesting session setup (AndX)
Requesting tree connect (AndX)
Requesting transaction (nuking) #1
Requesting transaction (nuking) #2
Requesting transaction (nuking) #3
Requesting transaction (nuking) #4
Requesting transaction (nuking) #5
Requesting transaction (nuking) #6
Requesting transaction (nuking) #7
Requesting transaction (nuking) #8
Requesting transaction (nuking) #9
Requesting transaction (nuking) #10
Wait...
Timeout during TCP read - Seems like the remote host has crashed
eve:~ $ ps –ef|grep dhcp
root       127     1  0 07:57 ?        00:00:00 /sbin/dhcpcd -d eth0
eve        354   264  1 08:39 pts/0    00:00:00 grep dhcp
```

```
eve:~ $ sudo kill 127
eve:~ $ sudo ifconfig eth0 10.21.5.188 broadcast 10.21.5.255 netmask 255.255.255.0 hw
ether 00:02:2d:38:83:2c down
eve:~ $ sudo ifconfig eth0 up
eve:~ $ sudo route add default gw 10.21.5.20
eve:~ $
```

4. With Vic's computer rebooting, Eve has the opportunity to access network resources, bypassing the WLAN security appliance. Eve could use this opportunity to access internal or external network resources, but only for a short time (until Vic's computer recovers). The duration between launching the DoS attack and Vic returning to an operational state is more than enough time to adequately scan and execute a buffer-overflow attack if automated with a scripting language. In this example, Eve simply connects to google.com and requests a HTTP GET.

```
eve:~ $ nc -vv www.google.com 80
www.google.com [216.239.39.101] 80 (http) open
GET / HTTP/1.0

HTTP/1.0 200 OK
Content-Length: 2532
Connection: Close
Server: GWS/2.0
Date: Fri, 15 Nov 2002 20:24:19 GMT
Content-Type: text/html
Cache-control: private
Set-Cookie: PREF=ID=26ac6f62286e5955:TM=1037391859:LM=1037391859:S=rCOeHto3hiJMX2gt;
expires=Sun, 17-Jan-2038 19:14:07 GMT; path=/; domain=.google.com

<html><head>
<snip>
eve: ~ $
```

Through sequence number analysis, we can establish a pattern of activity between Eve and Vic, as demonstrated in the following Ethereal extract summary:

```
IEEE 802.11
    Type/Subtype: Data (32)
    Destination address: 00:02:2d:09:a1:dd (00:02:2d:09:a1:dd)
    Source address: 00:02:2d:38:83:2c (00:02:2d:38:83:2c)
    Fragment number: 0
    Sequence number: 2378
Internet Protocol, Src Addr: 10.21.5.188 (10.21.5.188), Dst Addr:
10.21.5.209 (10.21.5.209)
Transmission Control Protocol, Src Port: 139 (139), Dst Port: 32930
(32930), Seq: 1204935225, Ack: 2064771556, Len: 92

IEEE 802.11
    Type/Subtype: Data (32)
    Source address: 00:02:2d:09:a1:dd (00:02:2d:09:a1:dd)
    Destination address: 00:02:2d:38:83:2c (00:02:2d:38:83:2c)
    Fragment number: 0
    Sequence number: 57
Internet Protocol, Src Addr: 10.21.5.209 (10.21.5.209), Dst Addr:
10.21.5.188 (10.21.5.188)
Transmission Control Protocol, Src Port: 32930 (32930), Dst Port:
139 (139), Seq: 2064771556, Ack: 1204935317, Len: 72

IEEE 802.11
    Type/Subtype: Data (32)
    Destination address: 00:02:2d:38:83:2c (00:02:2d:38:83:2c)
    Source address: 00:02:2d:09:a1:dd (00:02:2d:09:a1:dd)
    Fragment number: 0
    Sequence number: 58
Internet Protocol, Src Addr: 10.21.5.209 (10.21.5.209), Dst Addr:
```

```
10.21.5.188 (10.21.5.188)
Transmission Control Protocol, Src Port: 32930 (32930), Dst Port:
139 (139), Seq: 2064771556, Ack: 1204935317, Len: 72

IEEE 802.11
    Type/Subtype: Data (32)
    Frame Control: 0x0108
    Source address: 00:02:2d:38:83:2c (00:02:2d:38:83:2c)
    Destination address: 00:02:2d:09:a1:dd (00:02:2d:09:a1:dd)
    Fragment number: 0
    Sequence number: 2379
Internet Protocol, Src Addr: 10.21.5.188 (10.21.5.188), Dst Addr:
10.21.5.209 (10.21.5.209)
Transmission Control Protocol, Src Port: 139 (139), Dst Port: 32930
(32930), Seq: 1204935317, Ack: 2064771628, Len: 50

IEEE 802.11
    Type/Subtype: Data (32)
    Destination address: 00:02:2d:09:a1:dd (00:02:2d:09:a1:dd)
    Source address: 00:02:2d:38:83:2c (00:02:2d:38:83:2c)
    Fragment number: 0
    Sequence number: 2380
Internet Protocol, Src Addr: 10.21.5.188 (10.21.5.188), Dst Addr:
10.21.5.209 (10.21.5.209)
Transmission Control Protocol, Src Port: 139 (139), Dst Port: 32930
(32930), Seq: 1204935317, Ack: 2064771628, Len: 50
```

In this trace we see a short TCP data exchange between Eve and Vic. With this information we can start to baseline the 802.11 sequence numbers in use by each client, identifying Vic with sequence numbers in the range 2378-2380 while Eve is using sequence numbers in the range 57-58.

In this trace, we see Vic making a connection to google.com following the previous NetBIOS exchange:

```
IEEE 802.11
    Type/Subtype: Data (32)
    Source address: 00:02:2d:38:83:2c (00:02:2d:38:83:2c)
    Destination address: 00:03:47:df:12:72 (00:03:47:df:12:72)
    Fragment number: 0
    Sequence number: 62
Internet Protocol, Src Addr: 10.21.5.188 (10.21.5.188), Dst Addr: 216.239.39.101 (216.239.39.101)
Transmission Control Protocol, Src Port: 32931 (32931), Dst Port: 80 (80), Seq: 2053575733, Ack:
0, Len: 0
    Flags: 0x0002 (SYN)

IEEE 802.11
    Type/Subtype: Data (32)
    Destination address: 00:02:2d:38:83:2c (00:02:2d:38:83:2c)
    Source address: 00:03:47:df:12:72 (00:03:47:df:12:72)
    Fragment number: 0
    Sequence number: 3841
Internet Protocol, Src Addr: 216.239.39.101 (216.239.39.101), Dst Addr: 10.21.5.188 (10.21.5.188)
Transmission Control Protocol, Src Port: 80 (80), Dst Port: 32931 (32931), Seq: 2064907013, Ack:
2053575734, Len: 0
    Flags: 0x0012 (SYN, ACK)

IEEE 802.11
    Type/Subtype: Data (32)
    Source address: 00:02:2d:38:83:2c (00:02:2d:38:83:2c)
    Destination address: 00:03:47:df:12:72 (00:03:47:df:12:72)
    Fragment number: 0
    Sequence number: 63
Internet Protocol, Src Addr: 10.21.5.188 (10.21.5.188), Dst Addr: 216.239.39.101 (216.239.39.101)
Transmission Control Protocol, Src Port: 32931 (32931), Dst Port: 80 (80), Seq: 2053575734, Ack:
2064907014, Len: 0
```

```
    Flags: 0x0010 (ACK)

IEEE 802.11
    Type/Subtype: Data (32)
    Source address: 00:02:2d:38:83:2c (00:02:2d:38:83:2c)
    Destination address: 00:03:47:df:12:72 (00:03:47:df:12:72)
    Fragment number: 0
    Sequence number: 64
Internet Protocol, Src Addr: 10.21.5.188 (10.21.5.188), Dst Addr: 216.239.39.101 (216.239.39.101)
Transmission Control Protocol, Src Port: 32931 (32931), Dst Port: 80 (80), Seq: 2053575734, Ack:
2064907014, Len: 15
    Flags: 0x0018 (PSH, ACK)
Hypertext Transfer Protocol
    GET / HTTP/1.0\n

IEEE 802.11
    Type/Subtype: Data (32)
    Destination address: 00:02:2d:38:83:2c (00:02:2d:38:83:2c)
    Source address: 00:03:47:df:12:72 (00:03:47:df:12:72)
    Fragment number: 0
    Sequence number: 3842
Internet Protocol, Src Addr: 216.239.39.101 (216.239.39.101), Dst Addr: 10.21.5.188 (10.21.5.188)
Transmission Control Protocol, Src Port: 80 (80), Dst Port: 32931 (32931), Seq: 2064907014, Ack:
2053575749, Len: 0
    Flags: 0x0010 (ACK)

IEEE 802.11
    Type/Subtype: Data (32)
    BSS Id: 00:01:f4:ec:08:e8 (00:01:f4:ec:08:e8)
    Source address: 00:02:2d:38:83:2c (00:02:2d:38:83:2c)
    Destination address: 00:03:47:df:12:72 (00:03:47:df:12:72)
    Fragment number: 0
    Sequence number: 65
Internet Protocol, Src Addr: 10.21.5.188 (10.21.5.188), Dst Addr: 216.239.39.101 (216.239.39.101)
Transmission Control Protocol, Src Port: 32931 (32931), Dst Port: 80 (80), Seq: 2053575749, Ack:
2064907014, Len: 1
    Flags: 0x0018 (PSH, ACK)
Hypertext Transfer Protocol
    \n

IEEE 802.11
    Type/Subtype: Data (32)
    Destination address: 00:02:2d:38:83:2c (00:02:2d:38:83:2c)
    Source address: 00:03:47:df:12:72 (00:03:47:df:12:72)
    Fragment number: 0
    Sequence number: 3843
Internet Protocol, Src Addr: 216.239.39.101 (216.239.39.101), Dst Addr: 10.21.5.188 (10.21.5.188)
Transmission Control Protocol, Src Port: 80 (80), Dst Port: 32931 (32931), Seq: 2064907014, Ack:
2053575750, Len: 1288
    Flags: 0x0018 (PSH, ACK)
Hypertext Transfer Protocol
    HTTP/1.0 200 OK\r\n
    Content-Length: 2532\r\n
    Connection: Close\r\n
    Server: GWS/2.0\r\n
    Date: Fri, 15 Nov 2002 19:27:34 GMT\r\n
    Content-Type: text/html\r\n
    Cache-control: private\r\n
    Set-Cookie: PREF=ID=72a042234033edf4:TM=1037388454:LM=1037388454:S=MSjrdBlnS0y9yBn0;
expires=Sun, 17-Jan-2038 19:14:07 GMT; path=/; domain=.google.com\r\n
    \r\n
    Data (973 bytes)
```

After examining this trace more closely however, we see that the 802.11 sequence numbers for Vic's MAC address ("00:02:2d:38:83:2c") are not in sequence with the previously established baseline. Rather, the sequence number pattern more closely reflects that of Eve in the range 62-

65. With this information, we can match the activity from the MAC address "00:02:2d:38:83:2c" as spoofed MAC activity from Eve.

Problems with Sequence Number Analysis

In order to identify out-of-order sequence numbers as having originated from a spoofed MAC address, we need to establish a baseline of monitored MAC addresses and the sequence numbers in use. With a fixed access point and fixed IDS appliance (or IDS code on an access point), we can reliably track sequence numbers for the access point by capturing all wireless traffic in the area. If we wish to monitor client MAC addresses and their sequence numbers however, we must be sensitive to roaming clients leaving the range of monitoring by the layer 2 wireless IDS.

If the layer 2 anomaly-based intrusion detection system tracks the sequence number values for a particular client and the client then roams out of the range of the IDS, we are no longer able to monitor sequence number patterns. When the client returns within range of the IDS, the sequence numbers that follow will appear anomalous due to the large gap in sequence values. In order to avoid this potential false-positive detect, the IDS must be sensitive to time delays when establishing patterns in sequence numbers. If the IDS detects a delay of more than a few seconds between receiving frames for the activity of a client, it must assume the client has roamed out of range or has reset their card and therefore must invalidate the tracked sequence number pattern.

It is also important to note that sequence numbers will sometimes skip values when changing the transmit/receive channel, or if the firmware should drop a malformed frame. A sequence number will also be reset to zero if the WLAN card is reinitialized by the PCMCIA bus or other controlling subsystem. For this reason, WLAN anomaly detection systems should not alert on a single missed sequence number or by a small gap in sequence numbers. In most cases, an attacker who is spoofing the MAC address of another client or access point will be significantly different than the legitimate source MAC. A rule of thumb for alerting might be a sequence number that appears out of order with a difference of plus or minus thirty sequence values.

While testing the sequence analysis patterns presented in this paper, I discovered that Lucent 802.11b cards, using all firmware releases to the present 8.10, do not follow the 802.11 specification for sequence number generation. These cards will send frames with low-numbered sequence values in an initially incremental fashion and will sporadically jump from their initial values to the sequence numbers used by the wireless access point. The card will mirror the sequence numbers used by the wireless access point for a short period of time before returning to it's initial sequence number pattern. Without alternate accommodation, this peculiar activity would generate a false positive detect for a spoofed MAC address when following the techniques detailed in this paper. A wireless intrusion analysis system needs to monitor the sequence numbers of the access point in the network as well as the monitored client sequence numbers for variations in the sequence number pattern; first comparing an anomalous sequence number value to the pattern presently in use by the wireless access point before generating an alert.

Future Developments

With the LibRadiate tool suite, attackers have a simple interface in which to gather and transmit custom 802.11 frames. Using LibRadiate, attackers have started testing the responses from access points and wireless clients using out-of-spec parameters in management frames. With a simple packet injection framework for Linux, we are certain to see a rise in wireless-specific attack tools.

With much of the 802.11 management framework reserved for future development, we can also expect to see activity from attackers looking for unique responses from crafted frames in an attempt to "fingerprint" wireless access points. Such activity might provide an attacker with enough information to uniquely identify wireless access points by manufacturer and software revision, allowing them to restrict their attack to only those exploits applicable to the target network.

Finally, the ability to control the functions handled by the firmware of wireless cards is being actively researched, both in an attempt to mitigate the detection techniques described in this paper and to remove the restrictions imposed by system firmware transmitting malformed management and control frames. For reasons including the reverse engineering of firmware provided by vendors, it is foreseeable that 802.11 "attack cards" become commonplace, designed specifically for use in launching denial-of-service or man-in-the-middle attacks against WLANs.

Conclusion

Through careful analysis of layer 2 WLAN activity with anomaly-based intrusion detection tools that are capable of analyzing data-link layer traffic, an administrator can detect applications that hide their activity (through changing MAC addresses) or attempt to masquerade traffic as coming from a legitimate source MAC.

In order to mitigate these likely developments in wireless network attack techniques, hardware manufacturers should consider adding intrusion analysis functionality to their access points. Such development may not be applicable for SOHO or low-cost equipment, such as those produced by Linksys or D-Link, but seems appropriate for manufacturers of enterprise-level equipment, such as Cisco or Enterasys.

The deployment of wireless IDS appliances would be a costly and cumbersome challenge for complex WLAN installations, possibly requiring a 1:1 ratio for IDSs to access points. Such a deployment would increase ROI time to the point where deployment of wireless LANs may be cost-prohibitive for many organizations. The option to upgrade existing APs to add intrusion detection functionality would therefore be attractive to many customers. Those vendors who have had the forethought to include extra memory and processing power in their access points will be well positioned to provide the intrusion detection functionality that would deliver a competitive advantage.

As more applications are developed to utilize the mobility offered by wireless networks, companies assume increased risk with the security and reliability of their applications. Network

administrators and intrusion analysts need to be aware of the risks associated with 802.11 network deployment, and the techniques that can be used to identify malicious client activity.

Works Cited

AirJack. "Advanced 802.11 Attack Tools." URL: http://802.11ninja.net/ (12 Nov 2002).

FakeAP. "Black Alchemy Weapons Lab." URL: http://www.blackalchemy.to/project/fakeap/ (19 Dec 2002).

IEEE. *"IEEE OUI and Company_id Assignments."* URL: http://standards.ieee.org/regauth/oui/oui.txt (13 Nov 2002).

Malinen, Jouni. "H*ost AP driver for Intersil Prism2/2.5/3."* File: README.prism2, URL: http://hostap.epitest.fi/ (13 Nov 2002).

Schiffman, Mike. "*Radiate 802.11b frame handling*." URL: http://www.packetfactory.net/projects/radiate/ (13 Nov 2002).

Wellenreiter. *"Wireless LAN Discovery and Auditing Tool"* URL: http://www.remote-exploit.org/ (19 Dec 2002).

Appendix A

```
#!/usr/bin/perl -w
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# "Doing linear scans over an associative array is like trying to club
#  someone to death with a loaded uzi."  --Larry Wall  (Thanks Larry)
#
# Net::Pcap code adapted from example code at remote-exploit.org (Thanks Max)
#
# You will need the Net::Pcap CPAN module installed to use this program.  Try
#  running ‘ # perl –MCPAN –e “install Net::Pcap” ‘ to install.
#
# MAC Addresses I Don't Want To See - 11/12/2002 Joshua.Wright@jwu.edu
# The most recent copy of this script is available at
#  http://home.jwu.edu/jwright/code/maidwts.pl


use strict;
use Net::Pcap;
use Getopt::Long;
use IO::Handle;

my (
   $pcap_err,          $opt_int,
   $pcap_descr,        $progname,
   $fullmac,           $opt_verbose,
   $opt_count,         $frameoui,
   $ieeeoui,           $manuf,
   $opt_help,          $opt_nopromisc,
   $opt_timeout,       $opt_filename,
   $headeroffset,      $opt_80211capture,
   $opt_8023capture,
   $READTIMEOUT,       $PROMISC,
   $SNAPLEN,           $TIMEOUT,
   );

my (
   %ouihash,
   );


$READTIMEOUT=1000;     # in ms
$PROMISC=0;
$SNAPLEN=12;

$progname = $0;
$progname =~ s,.*/,,;    # only basename left in progname
$progname =~ s/\.\w*$//; # strip extension if any


# Main

Getopt::Long::Configure ("bundling_override");
GetOptions(
  'interface|i=s',      \$opt_int,
  'help|h',             \$opt_help,
  'count|c=i',          \$opt_count,
  'timeout|t=i',        \$opt_timeout,
  'nopromisc|n',       \$opt_nopromisc,
  'verbose|v',         \$opt_verbose,
```

```
  'filename|f=s',     \$opt_filename,
  'rfmonwlan|a',      \$opt_80211capture,
  'stdethernet|z',    \$opt_8023capture,
  ) or Usage('');


Usage('') if ($opt_help);
$opt_count = -1 unless($opt_count);
$opt_filename = "oui.txt" unless ($opt_filename);
$PROMISC = 1 unless ($opt_nopromisc);
$READTIMEOUT = $opt_timeout if ($opt_timeout);
Usage("Must specify capture interface with -i") unless ($opt_int);

# When capturing raw 802.11 frames, the first 4 bytes are for the Type/Subtype,
# frame control and duration fields.  When not capturing in RFMON, the std
# ethernet frame header applies.  This could probably be simplified in a
# $var1 ? $var2 : $var3 statement, but I think this is easier to read.
if ($opt_80211capture) {
  $headeroffset = 4;
  $SNAPLEN = ($SNAPLEN + $headeroffset);
} else {
  $headeroffset = 0;
}


# Trap CTRL/C to exit gracefully
$SIG{INT} = \&sigint_handler;


# Flush STDOUT
$|=1;

# Open the list of allocated OUIs

print "Reading list of OUI allocations from $opt_filename .. ";
if (! (open(OUILIST, "<$opt_filename"))) {
  warn "Unable to open file: $! \n";
  print "You will need an up to date copy of the IEEE OUI allocations.\n";
  print "You can download a copy from http://standards.ieee.org/regauth/oui/oui.txt\n";
  exit(1);
}
print "Done.\n";

# Slurp in list of OUI prefixes and assigned manufacturers, building hash
print "Generating hash .. ";
while (<OUILIST>) {
  if (/base 16/) {
    s/\(base 16\)//;
    ($ieeeoui, $manuf) = split(/\t\t/,$_);
    chomp $manuf;
    $ieeeoui =~ s/\s+//;
    $ieeeoui =~ uc($ieeeoui);
    # Some of the entries in the IEEE allocation file have a blank line next
    # to the base 16 representation of the allocation.  The preceding line
    # is labeled "PRIVATE", so we match that information here.
    unless ($manuf =~ /[A-Za-z]/) {
      $manuf="PRIVATE";
    }
    $ouihash{$ieeeoui} = $manuf;
  }
}
print "Done.\n";


# Open capture interface and start looping on each packet captured
print "Capturing traffic .. \n";
if ($pcap_descr = Net::Pcap::open_live
              ($opt_int,$SNAPLEN,$PROMISC,$READTIMEOUT,\$pcap_err)) {
  Net::Pcap::loop($pcap_descr, $opt_count, \&process_pkt, '' );
  } else {
  print "Could not open device \'$opt_int\' for capture: $pcap_err\n";
```

```
  exit(1);
}

# opt_count = 0
print "\nDone.\n";

# close pcap
if ($pcap_descr) {
  Net::Pcap::close ($pcap_descr);
}

exit(0);


sub process_pkt {
  my ($data, $header, $packet) = @_;
  $fullmac = unpack('H*', substr($packet, (6 + $headeroffset), 6));
  unless ($opt_count eq -1) {
    $opt_count--;
  }
  $frameoui = substr($fullmac,0,6);
  $frameoui = uc($frameoui);

  if (exists($ouihash{$frameoui})) {
    if ($opt_verbose) {
      print "Prefix $frameoui assigned to $ouihash{$frameoui}\n";
    }
  } else {
    print "Alert: Unallocated OUI prefix for address $fullmac\n";
  }
}

sub sigint_handler {
  # Close pcap gracefully after CTRL/C
  if ($pcap_descr) {
    Net::Pcap::close ($pcap_descr);
  print "\n";
    exit(0);
  }
}

sub Usage {
  my($msg) = @_;
  select(STDERR);
  autoflush STDERR 1;
  print STDERR "$progname: $msg\n" unless ($msg eq "");
  print "Usage:\n";
  print "  $progname [options]\n";
  print "    -i, --interface\n";
  print "    -f, --filename\n";
  print "    -c, --count\n";
  print "    -n, --nopromisc\n";
  print "    -t, --timeout\n";
  print "    -a, --rfmonwlan\n";
  print "    -z, --stdethernet\n";
  print "    -v, --verbose\n";
  print "    -h, --help\n";
  print "\n";
  print "e.x. \"$progname -c 500 -i eth1 -a\" To capture in 802.11 RFMON\n";
  print "e.x. \"$progname -c 500 -i eth1 -z\" To capture std ethernet frames\n";
  exit(1);
}
```