Promiscuous Host and Rudimentary Intrusion Detection with Pcap and Libnet

Ian Will

December 10, 2010

Abstract

Use of open or unsecure wireless networks, such as those frequently available in coffee shops, is increasingly a predominant method of accessing the internet. In these settings, computers are not shielded by a hardware firewall as is typical in home or corporate networks. In these settings, proper evaluation of the risk involved in using that network is desirable. To help evaluate risk, abnormal behavior must be identified. The goal of this project is to develop software that identifies abnormal behavior for public-access networks and elevates visibility to system users.

Introduction

In an open wireless network situation, there are two primary security concerns: other hosts engaged in traffic sniffing; and viruses, worms, or other forms of direct interaction with the machine. To provide visibility of these two potential events, the software should attempt to identify hosts in promiscuous mode, and direct communication from peers on the local network. The goals were both accomplished through a C based program which relies on libpcap for packet capture and libnet for packet generation.

The project was implemented as a C-based command line tool called Open Wireless Access Listener (OWAL) using libpcap for packet capture and libnet for packet generation. It runs two primary functions, the first is capturing traffic and inspecting packet headers for sources within the local network. The second is generating ARP probes to test for network interfaces in promiscuous mode. The traffic capturing function also identifies ARP responses to generated probes. In either of the two anomolous case, warnings are printed to the terminal.

Steps to Build

```
vicious:networks-ids ian$ make
gcc -g wireless-listener.c -o owal -I../libpcap -L../libpcap
-lpcap -lnet
vicious:networks-ids ian$
```

Installation

The program was developed and tested under Mac OS X. No OS X specific calls were made, so the program should work on Linux without modification but this was not tested. Owal does rely on unistd.h for command line argument parsing, so will not compile on Windows without modification. Owal relies on libpcap and libnet, which must be compiled and linked against when building owal. The source for those libraries is included with the assignment, but each must be built separately. I had no problems building libpcap, but libnet proved troublesome. Eventually I found that the libnet package was available via the homebrew package manager for OS X and used that. I've included the source for the libnet library I compiled with, but I was not able to compile it on OS X. I recommend using the homebrew version.

Running the Program

The typical method of running owal is

```
sudo ./owal -d en1 -p
```

which runs owal on en1 (the wireless network interface on OS X) in promiscuous mode. Other options are available, including specifying a BPF filter which reduces the number of packets passed through to owal, allowing control of false alarms and performance. The full list of arguments is available via -h, reproduced below.

```
vicious:networks-ids ian$ ./owal -h
./owal - Open Wireless Access Listening
Usage:  ./owal [-p] [-f filter] [-d device] [-t timeout]
               [-b probe sleep] [-l link type]
        -p      listen in promiscuous mode
        -h      print help and exit
        -f      pcap filter expression
        -d      specify the device to listen on
        -t      specify the packet capture timeout in ms,
                value is 1000 if unspecified
        -b      specify the sleep time between ARP probes
                in seconds, value is 3 if unspecified
        -l      specify the data link type, typicall one of
                EN10MB, IEEE802_11_RADIO, IEEE802_11, or
                IEEE802_11_RADIO_AVS
```

Owal must be run as root for ARP packet construction. Typically libpcap also requires elevated privileges, though in pcap's case there is a workaround (a system startup script that changes device privileges). I could find no such workaround for libnet.

Also note that although the -d option to specify the device is optional, pcap is poor at automatically selecting an appropriate interface on Mac OS X, so it must be specified explicitly.

As it runs owal monitors network traffic, generating ARP probes for newly discovered nodes and characterizing peers based on the determination of promiscuous state and whether they are directly communicating with this host. Peers that are characterized as concerning result in warnings printed to the console. Owal continues monitoring traffic and generating ARP probes for new peers until it is closed. To stop owal, use ctrl-c. An example of the output while running is below.

```
vicious:networks-ids ian$ sudo ./owal -d en1 -p
Available data link types are...
1 - EN10MB - Ethernet
7f - IEEE802_11_RADIO - 802.11 plus radiotap header
69 - IEEE802_11 - 802.11
a3 - IEEE802_11_RADIO_AVS - 802.11 plus AVS radio information header
Set data link to  EN10MB: Ethernet.
...done with initialization
--------------------------------------------------
Network address:        192.168.1.0
Network mask:           255.255.255.0
Gateway:                192.168.1.1
Interface name:         en1
MAC address:            78:ca:39:b8:ac:c1
IP address:             192.168.1.122
Beginning monitoring...(Ctrl-C to quit)
--------------------------------------------------
Probing 192.168.1.99...
        with ff:ff:ff:ff:ff:fe...
        with ff:ff:00:00:00:00...
        ARP reply from 00:19:e3:d3:f5:7a/Apple Comput (192.168.1.89)
        ARP reply from 00:19:e3:d3:f5:7a/Apple Comput (192.168.1.89)
        with ff:00:00:00:00:00...
        ARP reply from 00:19:e3:d3:f5:7a/Apple Comput (192.168.1.89)
        ARP reply from 00:19:e3:d3:f5:7a/Apple Comput (192.168.1.89)
        with 01:00:00:00:00:00...
        ARP reply from 00:19:e3:d3:f5:7a/Apple Comput (192.168.1.89)
        ARP reply from 00:19:e3:d3:f5:7a/Apple Comput (192.168.1.89)
        with 01:00:5e:00:00:00...
        ARP reply from 00:19:e3:d3:f5:7a/Apple Comput (192.168.1.89)
        ARP reply from 00:19:e3:d3:f5:7a/Apple Comput (192.168.1.89)
        with 01:00:5e:00:00:01...
        ARP reply from 00:19:e3:d3:f5:7a/Apple Comput (192.168.1.89)
        ARP reply from 00:19:e3:d3:f5:7a/Apple Comput (192.168.1.89)
Probing 192.168.1.89...
        with ff:ff:ff:ff:ff:fe...
        ARP reply from 00:19:e3:d3:f5:7a/Apple Comput (192.168.1.89)
Warning:  host 192.168.1.89(00:19:e3:d3:f5:7a/Apple Comput) appears to be in promiscuou
        ARP reply from 00:19:e3:d3:f5:7a/Apple Comput (192.168.1.89)
        with ff:ff:00:00:00:00...
        ARP reply from 00:19:e3:d3:f5:7a/Apple Comput (192.168.1.89)
        ARP reply from 00:19:e3:d3:f5:7a/Apple Comput (192.168.1.89)
```

```
...
^C
Capture stopped.
vicious:networks-ids ian$
```

Architectural Design

The program is broken into three basic functions, which correspond to three C header files:

1. Gathering information about the host and interface being monitored (owal_info.h)
2. Packet capture and analysis (owal_callback.h)
3. ARP probe generation (probe_arp.h)

The initial phase of execution is the information gathering phase, which puts together information about the host and interface into a C structure.

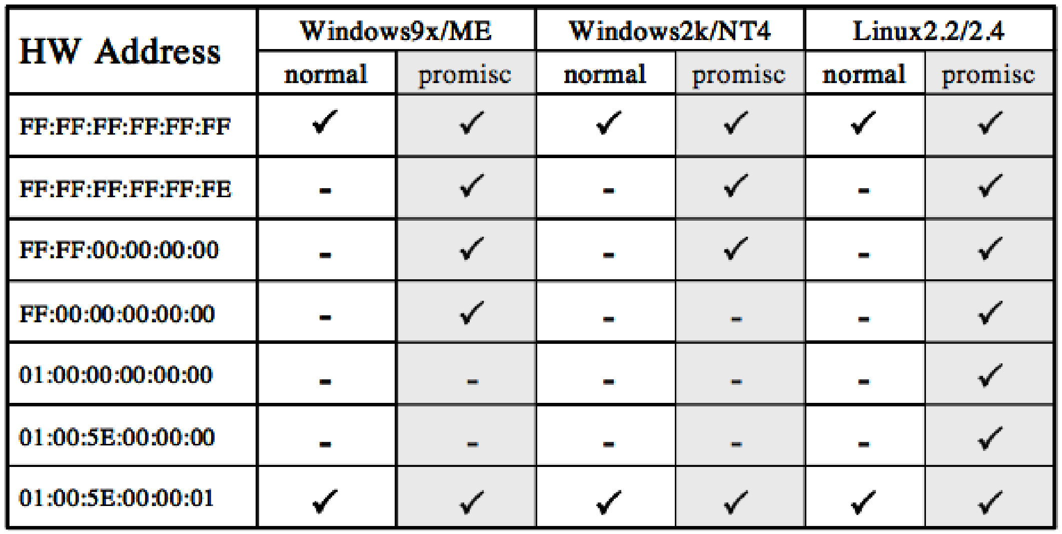
```
struct owal_if_info {
  bpf_u_int32 net;
  bpf_u_int32 mask;
  bpf_u_int32 broadcast;
  bpf_u_int32 gateway;
  bpf_u_int32 ip;
  u_int8_t mac[6];
  char* name;
  int link_type;
};

typedef struct owal_if_info * owal_if_info_p;
```

After this information is gathered, two threads are created (using the pthread library), one runs the packet capture loop and the other runs ARP probe creation. The two threads communicate via four shared lists of ip addresses. As the packet capture thread identifies new network hosts, it places them in a gray list, which the ARP probe thread watches. Each ip address in the gray list is probed for promiscuous configuration and moved to the probed list. The packet capture thread then watches for ARP responses directed to this machine, and if they come from an ip address in the probed list, the host may be in promiscuous mode. In this case, the ip address is moved into a black list and a warning is printed to the console. A white list is not used in this implementation, but could be used in the future along with a timeout for probes to prevent mischaracterizing hosts which ignore ARP probes but later respond to legitimate ARP queries. The list structures and functions are implemented in ip_lists.h.

Probing for Promiscuous Hosts Using ARP

The technique used to probe for promiscuous hosts was based on an article from Security Friday [1]. The approach is to create an ARP packet querying an IP for its hardware address, however instead of setting the destination in the ethernet packet to FF:FF:FF:FF:FF:FF, it sets the destination to a non-existent address. Network interfaces in normal mode will reject the ethernet packet, but interfaces in promiscuous modes will accep it and respond. In this manner, nodes in promiscous mode can be identified. There are a few additional considerations based on the ways different operating systems react to these packets, figure 1 has a table listing responses to probes by address and operating system. Practically, probing FF:FF:FF:FF:FF:FE and FF:FF:00:00:00:00 yields the most consistent results across operating systems. Owal probes each of the addresses in the table.



HW Address	Windows9x/ME		Windows2k/NT4		Linux2.2/2.4	
	normal	promisc	normal	promisc	normal	promisc
FF:FF:FF:FF:FF:FF	✓	✓	✓	✓	✓	✓
FF:FF:FF:FF:FF:FE	-	✓	-	✓	-	✓
FF:FF:00:00:00:00	-	✓	-	✓	-	✓
FF:00:00:00:00:00	-	✓	-	-	-	✓
01:00:00:00:00:00	-	-	-	-	-	✓
01:00:5E:00:00:00	-	-	-	-	-	✓
01:00:5E:00:00:01	✓	✓	✓	✓	✓	✓

Figure 1: Responses to ARP probes by OS [1]

Detecting Unauthorized Access from Peers

As packets are captured, those which are sent to the running machine by network peers are treated as suspect and warnings are printed to the console. I found it was not unusual for peers to communicate directly, even in public wireless networks. TCP port 139 (NetBIOS session service), which is used for file and printer sharing, and as UDP port 137 (NetBIOS name service) were occasionally observed. For this reason, in addition to the warning about peer access, the protocol and port are also printed to provide some context. For software to be more useful to a typical user, those would need to be correlated with typical applications. TCP port 139 and UDP port 137 could be filtered out, as they are not particularly alarming behavior, but they were left in for the assignment as an easy way to demonstrate a working program and because excessive traffic over those ports may indicate nefarious purposes.

Correlating Hardware Address with Manufacturer

While running owal in local coffee shops, I was surprised at the number of hosts that responded to ARP probes. To lend more context to these responses, I found it was helpful to correlate the hardware MAC address with the name of the vendor as assigned by IEEE. Wireshark's correlation process uses associations maintained at [2], which is a cleaned up version of those available from IEEE. However, I found many vendor codes were missing, so I combined those with the codes from the IEEE-SA Registration Authority [3].

Many of the hosts that responded to ARP probes came from Apple manufactured network interfaces. This suggests a more thorough investigation into the behavior of OS X would be useful. OS X was not explicitly characterized in figure 1. It seems unlikely that so many Mac users had nefariously configured their computers in promiscuous mode to intercept traffic. It could be that Mac OS X treats ARP requests to non-broadcast hardware addresses differently than Windows and Linux.

Implementation Notes

1. Pcap can be run without sudo privileges with proper modification of device privileges. The process is described in the README.macosx file in the libpcap directory. It involves copying startup scripts to /Library/StartupItems/ that adds rw privileges to all bpf devices in /dev.

2. Libnet can not be run without sudo privileges. I found no work arounds.

3. There appears to be no way to find gateway information using libpcap or libnet, nor does there appear to be a good way to access this information via C. The best solution I could find parses the output of netstat. I used modified code from [4] to turn the output into a 32 bit integer. It pipes the output of netstat into a temporary file, then reads the results in and parses out the gateway. Knowing the gateway is important so normal traffic between the host and gateway can be identified and ignored.

4. Pcap doesn't seem to have a good interface for accessing the MAC address. With libnet this easy, using the libnet_get_hwaddr function.

5. Libnet is difficult to find on google, it was historically hosted at `www.packetfactory.net/libnet/`, which is no longer accessible. There is a sourceforge page [5], but it was last updated in 2003 and the version of libnet available there is version 0.10.11. The only active branch is hosted

at [6], it's at version 1.1.5. The openwall website [7] also hosts a few versions of libnet, including 1.0.2a (the last version released by the original maintainer), and up to 1.1.3 of the 1.1.x tree. Although the source is out of date, there are good documentation resources there, including a presentation on libnet by Mike Schiffman [8] at the RSA conference in 2004. It seems odd that libnet is not better supporrted, because many networking packages rely on it, including netwib/netwox/networks and snort.

References

[1] D. Sanai, Aug 2001. [Online]. Available: http://www.securityfriday.com/promiscuous_detection_01.pdf

[2] "Ethernet codes master list." [Online]. Available: http://www.cavebear.com/archive/cavebear/Ethernet/

[3] "IEEE-SA registration authority organizationally unique identifier (OUI)." [Online]. Available: http://standards.ieee.org/develop/regauth/oui/oui.txt

[4] tincuchalice, "Howto find gateway address through code," Dec 2005. [Online]. Available: http://www.linuxquestions.org/questions/linux-networking-3/howto-find-gateway-address-through-code-397078/

[5] "Libnet home page," Feb 2003. [Online]. Available: http://libnet.sourceforge.net/

[6] "sam-github's libnet branch," Nov 2010. [Online]. Available: https://github.com/sam-github/libnet

[7] M. Schiffman, "The million packet march," Feb 2007. [Online]. Available: http://packetfactory.openwall.net/projects/libnet

[8] ——, "The evolution of libnet," Feb 2004. [Online]. Available: http://packetfactory.openwall.net/projects/libnet/2004_RSA/eol-1.0.key.zip