A Simple Introduction To Writing Sniffers for 802.11 Networks

Vivek Ramachandran

I will assume the following :---

1. You have linux installed , preferabley the latest release
   with all drivers loaded & a wireless interface up and running .

2. Libpcap is installed .

3. You have a basic knowledge of C programming on the linux
   platform .

if you do not have any of the above then i guess reading ahead
would be of no use . If yes !! then welcome aboard :-) !!!
to the sniffer 101 course .

Lesson 1 : SSID sniffing

The ssid is the most fundamental thing required by a client (STA)
before it associates with a an Access Point (AP) .
The ssid is broadcasted in Beacon Frames by the AP , so that an
STA in it's range can locate it . Well such broadcasting can even
be turned off by the admin of the AP .....but we could still get it
....I'll come it that in lesson 2 .

So the program is going to work as follows :----

1. Initialise libpcap and make it sniff on an interface .

2. Libpcap will pass on the packets collected to our "decoding"
   function .

3. Our function checks if the packet is a Beacon Frame packet
   or not ?? If yes we get the ssid out of it .

4. We just loop it to probe for more ssids .

Coding :---

A general frame format for an 802.11 Management Frame is :---

Frame Control	Duration	DA	SA	BSSID	Seq cnt	Frame body	FCS
<---2 --->	<--- 2 --->	<-6->	<-6->	<---6--->	<--2->	<- var ->	<--4->

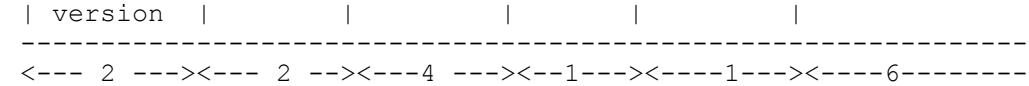
Note : All figures are in Octets . The frame body is of variable
       size depending upon what that frame carries . This will be
       clear in a minute .

DA - Destination Add
SA - Source Add
BSSID - Broadcast SSID

We proceed as follows :--

1. There are many subtypes of management frames of which we are
   interested in type Beacon frames which is broadcasted on a
   periodic basis by the AP .

2. The Frame control field decides if it's of management type
   Beacon . Here's how the frame field looks like ....



Note : All in figs in the above are in BITS . There is a lot of
info available in the frame control field , but for our
work we just need the type & subtype .

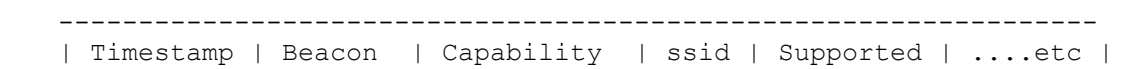
3. So once we recv a frame we check from protocol version that it is
indeed 802.11 frame , for which this field is 0 .

4. We check the type for type Management for which this field is 0 .

5. Then we check the management frame subtype for type Beacon frame
for which this filed is 0x08 .

6. Then if it is indeed Beacon we pass on to check the ssid from the
frame body field .

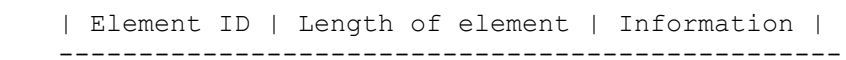
7. The frame body field for a beacon frame has the following structure



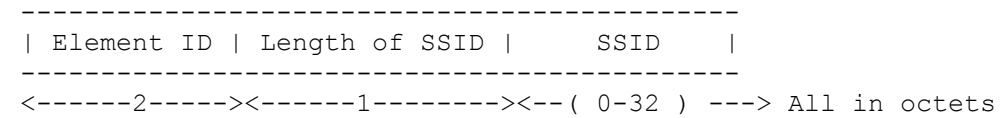
Note : All values in octets .

In the frame body we have fixed lengths and variable length fields
(called information fields in 802.11 terminology) . The timestamp,
beacon interval & capability info are fixed length and ssid etc are
the variable length ones . Why is the ssid variable length ?? coz it
just depends on what the admin chooses it to be .

As we are primarily interested in just finding the broadcasted ssid
so we'll forget about the other fields for the time being .
All information fields can be broken up as follows :---



For an ssid the above would transform to :



8. Now we would like to have our own data structure to hold all
this info , so here they are :

--------headers.h----------------------

```
typedef struct mac_header{
unsigned char fc[2];
unsigned char id[2];
unsigned char add1[6];
unsigned char add2[6];
unsigned char add3[6];
unsigned char sc[2];
}mac_header;

typedef struct frame_control{
unsigned protocol:2;
```

```
unsigned type:2;
unsigned subtype:4;
unsigned to_ds:1;
unsigned from_ds:1;
unsigned more_frag:1;
unsigned retry:1;
unsigned pwr_mgt:1;
unsigned more_data:1;
unsigned wep:1;
unsigned order:1;
}frame_control;

typedef struct beacon_header{
unsigned char timestamp[8];
unsigned char beacon_interval[2];
unsigned char cap_info[2];
}beacon_header;
```

Just looking at the ascii frame representations , the above data structures should be clear .

So here is the program now : ---

----------main.c----------

```
#include<stdio.h>
#include<stdlib.h>
#include<pcap.h>
#include<errno.h>
#include<arpa/inet.h>
#include<net/ethernet.h>
#include<linux/wireless.h>
#include<netinet/if_ether.h>
#include"headers.h"

void
packet_decoder (u_char * useless, const struct pcap_pkthdr *pkthdr,
            const u_char * packet)
{
  char ssid[32], *temp;
  struct mac_header *p = (struct mac_header *) packet;
  struct frame_control *control = (struct frame_control *) p->fc;
  temp =
    (char *) (packet + sizeof (struct mac_header) +
            sizeof (struct beacon_header));

  memset (ssid, '\0', 32);

// check if frame is beacon frame
  if ((control->protocol == 0) && (control->type == 0)
      && (control->subtype == 8))
    {
//temp[1] contains the size of the ssid field and temp[2] the beginning of
//the ssid string .

      memcpy (ssid, &temp[2], temp[1]);
      printf ("\n\nFound SSID  : \n");
      printf ("Destination Add : %s\n", ether_ntoa (p->add1));
      printf ("Source      Add : %s\n", ether_ntoa (p->add2));
      printf ("BSSID           : %s\n", ether_ntoa (p->add3));
      printf ("ssid = %s\n", ssid);
    }
}

int
main (int argc, char **argv)
{
  char *dev = argv[1];
```

```
  char errbuf[PCAP_ERRBUF_SIZE];
  pcap_t *descr;

  if (argc < 2)
    {
      printf ("usage : %s capture_device \n", argv[0]);
      exit (1);
    }
  printf ("\n\nSSID sniffer : \nInitialising capture interface ........");

  //pcap initialisation
  descr = pcap_open_live (dev, BUFSIZ, 1, -1, errbuf);
  if (descr == NULL)
    {
      printf ("pcap_open_live : %s\n", errbuf);
      exit (1);
    }
  printf ("\nStarting Capture ...........\n");
  // tell pcap to pass on captures frames to our packet_decoder fn
  pcap_loop (descr, -1, packet_decoder, NULL);

  return (0);

}
-------------------------------------------------------
```

You could just cut paste this code and compile it with :

```
gcc main.c -lpcap -o sniffer
```

I guess if you have got everything right than it should have compiled without any errors . So time to sniff !!!!

```
./sniffer wifi0
```

Notes :

I use wifi0 with my cisco aironet card . If all goes well you will be seeing a lotta activity with maybe the same ssid being displayed a lotta times . The reason is we have not implemented a checking if the ssid obtained currently was also obtained prevoiusly . I leave this trivial problem for you to solve .

Note that you may have to put the card into monitor mode before sniffing . You could do this manually or use something like kismet_monitor (it comes with the kismet sniffer) .

If ssid broadcastin has been turned off by the admin then the ssid field would appear blank . There are still ways to get the ssid which i will discuss in Lesson 2 .

The code written might not be the best , there can always be easier ways to get the job done . All i am showing is a proof of concept & the easiest code i could write . I invite people with better ways to send me a copy of theirs so that i may learn .

Lesson 2 : SSID collection from non broadcasting networks

Well , as i had previously mentioned some admins may disable ssid broadcasting . But forunately we could still sniff it .

The reason is as follows :----

Whenever an STA wants to associate with an AP then it probes for it with a Probe Request frame , to which the AP replies back with a Probe Response Frame . Unfortunately both these frames contain

the ssid in them and for very obvious reasons the response frame
is a better bet to what the ssid actually is .
As a point of note the Association Request frames also contains
the ssid.

I have added comments in the stealth.c file which shows how we
could check for these frames . Also note that all these frames
will be having the ssid field position different . So better
check up the standard before coding .

Note that all the ssid collection methods described above are
all passive which means we are just monitoring traffic and
not injecting anything into the stream . Well there are some
active methods of ssid detection too , used by some open
source tools but be warned that there already exists
IDS (Intrusion Detection Systems) , which can detect such
scans .

I guess Lesson 3 would be dealing with active methods of
obtaining ssid's .