Scapy BTBB Demo

- This demo serves as a brief scapy tutorial but more importantly, it illustrates the btbb layer in Scapy
- it also demonstrates utilities and helpers provided by the library
- if you have issues installing the btbb scapy module, please refer to the documentation at hackgnar.com

library imports

- import everything from scapy for the demo
- import everything from the btbb Scapy module

In [1]:
```
from scapy.all import *
from btbb import *
```

```
WARNING: No route found for IPv6 destination :: (no default route?)
WARNING:scapy.runtime:No route found for IPv6 destination :: (no default
route?)
```

Open btbb pcap file:

- btbb pcap files for this demo were created with Kismet and Ubertooth
-
    - these can also be created by other means such as USRP and Kismet, etc

In [2]:
```
btbb_pcaps = PcapReader('../../data/small.pcapbtbb')
```

Read one packet from the pcap file:

- btbb packet is read pcap file and instantiated as Scapy packet

In [3]:
```
pkt = btbb_pcaps.read_packet()
```

Packet sample:

- nothing special about this packet. Looks like a typical Ethernet packet
- btbb packets are layered on top of the ethernet layer much like the wireshark btbb layout
- when nothing is present in the btbb layer, these look exactly like ethernet packets

In [4]:
```
pkt.show()
```

```
###[ Ethernet ]###
  dst       = 00:00:00:00:00:00
  src       = 00:00:00:ed:1d:9c
  type      = 0xfff0
```

Interactively iterate through packets:

- we can run the following over and over to look though packets

In [5]:
```
pkt = btbb_pcaps.read_packet()
pkt.show()
```

...

In [6]:
```
pkt.summary()
```

Out[6]:
```
'00:00:00:ed:1d:9c > 00:00:00:00:00:00 (0xfff0)'
```

Conditionally iterate though btbb pcap file:

- iterate though the pcap file
- display summary data for all packets
- display detailed data if a btbb payload exists

In [7]:
```
for pkt in btbb_pcaps:
    print pkt.summary()
    if pkt.haslayer('BtbbPayload'):
        pkt.show()
        break
```

```
00:00:00:ed:1d:9c > 00:00:00:00:00:00 (0xfff0)
00:00:00:ed:1d:9c > 00:00:00:00:00:00 (0xfff0)
00:00:00:ed:1d:9c > 00:00:00:00:00:00 (0xfff0)
00:00:00:ed:1d:9c > 00:00:00:00:00:00 (0xfff0)
00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket / BtbbPayload
###[ Ethernet ]###
  dst       = 00:00:00:00:00:00
  src       = 00:00:36:ed:1d:9c
  type      = 0xfff0
###[ btbb ]###
###[ meta ]###
        CLK       = 0x7000000L
        Channel   = 39L
        Padding   = 0L
        known address bits= 32 (NAP unknown)
        known clock bits= 6
###[ packet ]###
           type      = DH1/2-DH1
           LT_ADDR   = 0x1L
           SEQN_Flag = 1L
           ARQN_Flag = 0L
           FLOW_Flag = 1L
           HEC       = 0xc9
###[ payload ]###
              header_length= 3L
              header_flow= 1L
              header_LLID= 0L
              body      = '\x0e\x13\xd1'
              CRC       = 0x6209L
```

Packet list

- instantiate the rest of the packets into a list of packets

In [8]:
```python
btbb_pkt_list = btbb_pcaps.read_all()
```

In [9]:
```python
btbb_pkt_list
```

Out[9]:
```
<small.pcapbtbb: TCP:0 UDP:0 ICMP:0 Other:456>
```

In [10]:
```python
print len(btbb_pkt_list)
for item in btbb_pkt_list[:5]:
    print item.summary()
```

```
456
00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
```

Write btbb pcap files:

- we can also write btbb packets back to new pcap files if we like

```
In [11]: pcapbtbb_writer = PcapWriter('../../data/new_pcap_file.pcapbtbb')
         pcapbtbb_writer.write(btbb_pkt_list)
```

```
In [12]: !ls -li new_pcap_file.pcapbtbb
```

```
ls: new_pcap_file.pcapbtbb: No such file or directory
```

```
In [13]: new_btbb_pkts = PcapReader("../../data/new_pcap_file.pcapbtbb")
         pkts = new_btbb_pkts.read_all()
         print len(pkts)
         for i in pkts[:5]:
             print i.summary()
```

```
412
00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
```

```
In [14]: new_btbb_pkts.close()
         btbb_pcaps.close()
```

Btbb Pcap File Stream:

- Generic way to stream data from bluetooth baseband hardware
- Relies on the fact that they have a way to write btbb pcap files
- Allows for interactive real time packet monitoring

```
In [15]: #log_dir = "../../data/new_pcap_file.pcapbtbb"
         #latest_file = !ls -t1 $log_dir|head -1
         #latest_file = log_dir + '/' + latest_file[0]
         latest_file = "../../data/new_pcap_file.pcapbtbb"
```

```
In [16]: btbb_stream = BtbbPcapStreamer(latest_file)
```

```
In [17]: for pkt in btbb_stream.stream(output='packet', stop=True):
             print pkt.summary()
```

```
00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
```

```
BtbbPacket
00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
```

In [18]:
```
btbb_stream.close()
```

Btbb layer helper methods

- a sample of some of the helper methods provided by scapy btbb
- lets open a new pcap file, read in the packets and define some vars first

In [19]:
```
manuf_file='../../data/wireshark_manuf'
!wc -l $manuf_file
```

```
   22118 ../../data/wireshark_manuf
```

In [20]:
```
btbb_pcaps = PcapReader('../../data/small.pcapbtbb')
pkts = btbb_pcaps.read_all()
```

In [21]:
```
for i in range(10):
    print i , pkts[i].summary()
```

```
0 00:00:00:ed:1d:9c > 00:00:00:00:00:00 (0xfff0)
1 00:00:00:ed:1d:9c > 00:00:00:00:00:00 (0xfff0)
2 00:00:00:ed:1d:9c > 00:00:00:00:00:00 (0xfff0)
3 00:00:00:ed:1d:9c > 00:00:00:00:00:00 (0xfff0)
4 00:00:00:ed:1d:9c > 00:00:00:00:00:00 (0xfff0)
5 00:00:00:ed:1d:9c > 00:00:00:00:00:00 (0xfff0)
6 00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket / BtbbPayload
7 00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
8 00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
9 00:00:36:ed:1d:9c > 00:00:00:00:00:00 (0xfff0) / Btbb / BtbbMeta /
BtbbPacket
```

Vendor lookup:

- can lookup vendor based on a bluetooth address
- can lookup vendor based on packet
- vendor determination is more accurate when both nap and uap are known
- when only a uap is know, a list of possible vendors and associated nap is returned
- if your wireshark manuf file is not in a default location you must specify as seen below

```
In [22]: get_vendor('00:11:36:ed:1d:9c', manuf_file=manuf_file)
```

```
Out[22]: [('00:11:36', 'Goodrich')]
```

```
In [23]: possible_vendors = get_vendor(pkts[6],manuf_file=manuf_file)
```

```
In [24]: len(possible_vendors)
```

```
Out[24]: 61
```

```
In [25]: possible_vendors
```

```
Out[25]: [('00:00:36', 'Atari'),
          ('00:01:36', 'Cybertan'),
          ('00:02:36', 'Init'),
          ('00:03:36', 'ZetesTec'),
          ('00:04:36', 'ElansatT'),
          ('00:05:36', 'DanamCom'),
          ('00:06:36', 'JedaiBro'),
          ('00:07:36', 'DataVide'),
          ('00:09:36', 'Ipetroni'),
          ('00:0A:36', 'SynelecT'),
          ('00:0B:36', 'Producti'),
          ('00:0C:36', 'SharpTak'),
          ('00:0D:36', 'WuHanRou'),
          ('00:0E:36', 'Heinesys'),
          ('00:0F:36', 'Accurate'),
          ('00:10:36', 'Inter'),
          ('00:11:36', 'Goodrich'),
          ('00:12:36', 'Consentr'),
          ('00:13:36', 'Tianjin7'),
          ('00:14:36', 'QwertyEl'),
          ('00:15:36', 'Powertec'),
          ('00:16:36', 'QuantaCo'),
          ('00:17:36', 'Iitron'),
          ('00:18:36', 'Reliance'),
          ('00:19:36', 'Sterlite'),
          ('00:1A:36', 'Aipermon'),
          ('00:1B:36', 'TsubataE'),
          ('00:1C:36', 'InewitNv'),
          ('00:1D:36', 'Electron'),
          ('00:1E:36', 'Ipte'),
          ('00:1F:36', 'BellwinI'),
          ('00:20:36', 'BmcSoftw'),
          ('00:21:36', 'Motorola'),
          ('00:22:36', 'VectorSp'),
          ('00:23:36', 'MetelSRO'),
          ('00:24:36', 'Apple'),
          ('00:25:36', 'OkiElect'),
          ('00:26:36', 'Motorola'),
          ('00:30:36', 'RmpElekt'),
          ('00:40:36', 'TribeCom'),
          ('00:50:36', 'Netcam'),
          ('00:60:36', 'AitAustr'),
          ('00:80:36', 'ReflexMa'),
          ('00:90:36', 'Ens'),
          ('00:A0:36', 'AppliedN'),
          ('00:C0:36', 'RaytechE'),
          ('00:D0:36', 'Technolo'),
          ('00:E0:36', 'Pioneer'),
          ('08:00:36', 'Intergra'),
          ('0C:E9:36', 'ElimosSr'),
          ('58:E6:36', 'EvrsafeT'),
          ('64:0E:36', 'Taztag'),
          ('68:5B:36', 'Powertec'),
          ('6C:83:36', 'SamsungE'),
          ('88:10:36', 'PanodicS'),
          ('8C:92:36', 'AusLinxT'),
          ('9C:4E:36', 'IntelCor'),
          ('AC:72:36', 'LexkingT'),
          ('B0:AA:36', 'Guangdon'),
          ('E0:26:36', 'NortelNe'),
          ('EC:F2:36', 'Neomonta')]
```

Distinct bluetooth address lookup:

- distinct bluetooth addresses can be looked up
- useful for quickly determining what devices are in a list of packets
- useful for passing to other tools/modules for analysis, exploitation, etc

In [26]:
```python
bt_addrs = get_btaddress(*pkts)
bt_addrs
```

Out[26]:
```
['00:00:00:c3:ec:46',
 '00:00:00:db:c1:fa',
 '00:00:36:ed:1d:9c',
 '00:00:d2:59:84:d9',
 '00:00:00:ff:c4:ab']
```

In [27]:
```python
import bluetooth
for addr in bt_addrs:
    print bluetooth.lookup_name(addr)
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-27-8b458bdeb67c> in <module>()
----> 1 import bluetooth
      2 for addr in bt_addrs:
      3     print bluetooth.lookup_name(addr)

ImportError: No module named bluetooth
```