Passive Aggressive Bluetooth Scanning

Ryan Holeman
@hackgnar

About Me

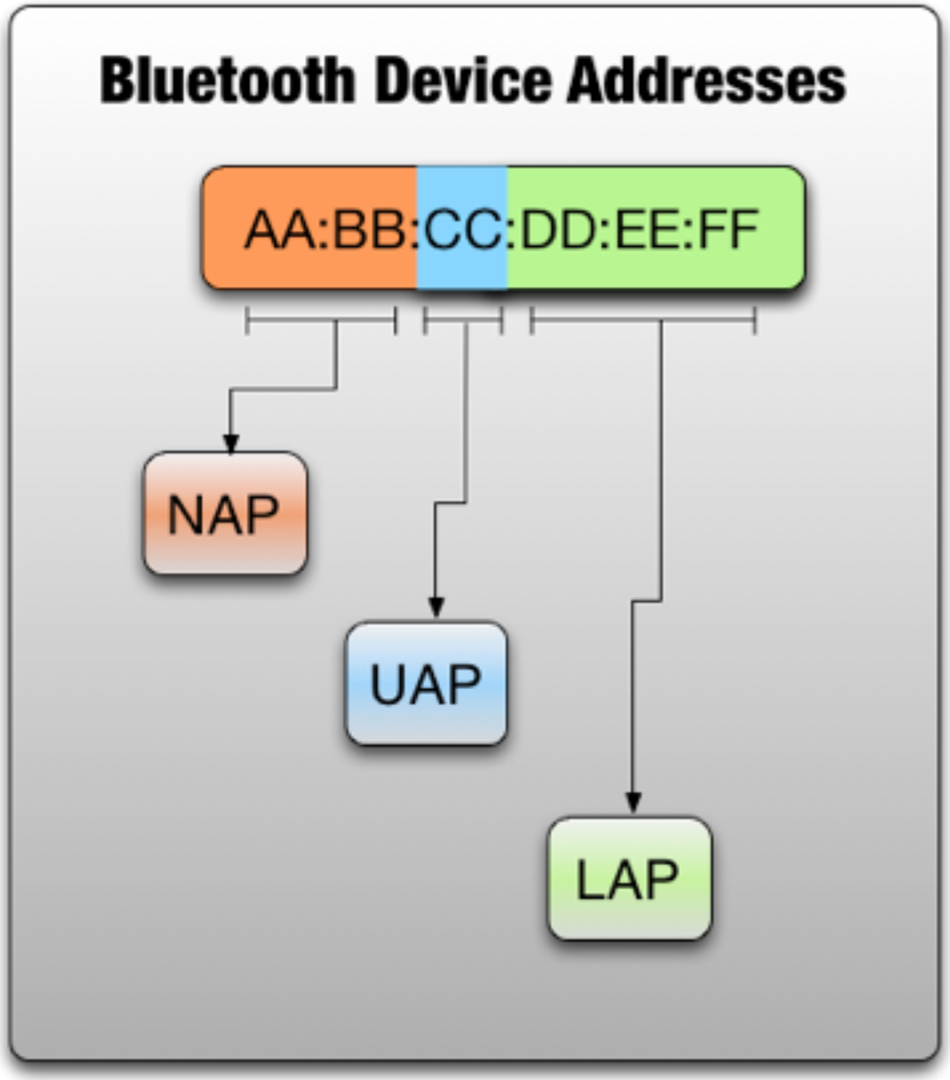
- Ziften Technologies
- Python & Ubertooth
- bluetoothdatabase.com
- github.com/hackgnar
- Twitter: @hackgnar

Agenda

- Active scan techniques
- Passive scan techniques
- Misc scan techniques
- Hybrid techniques

Bluetooth Address

- NAP & UAP
  - Vendor association
- LAP
  - Device specific

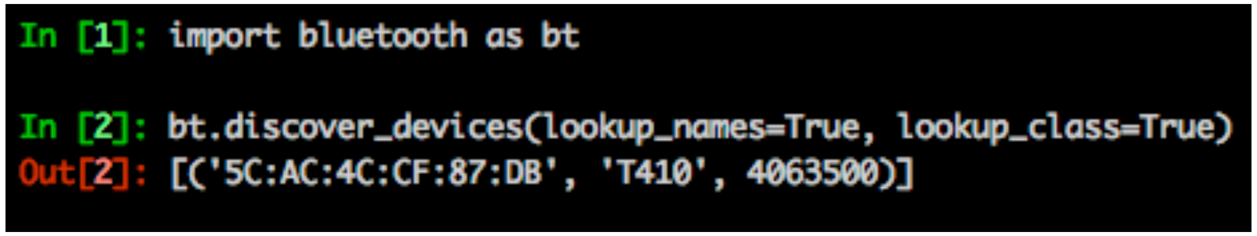


Active Scan Techniques

- Commodity bluetooth hardware
- Scanned devices typically need to be in discoverable mode
- Actively queries device
- Range is based on device class

Discovery Scan

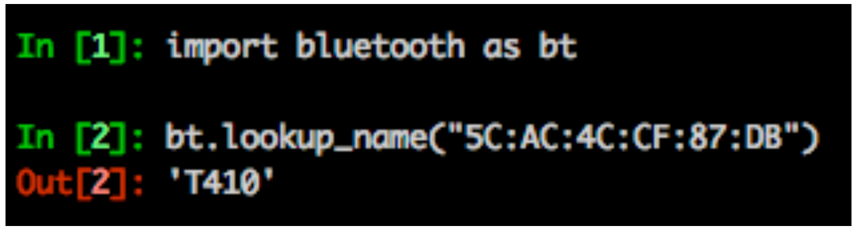
- Obtains
  - Address
  - Name
  - Class info



```
In [1]: import bluetooth as bt

In [2]: bt.discover_devices(lookup_names=True, lookup_class=True)
Out[2]: [('5C:AC:4C:CF:87:DB', 'T410', 4063500)]
```

Name Inquiry

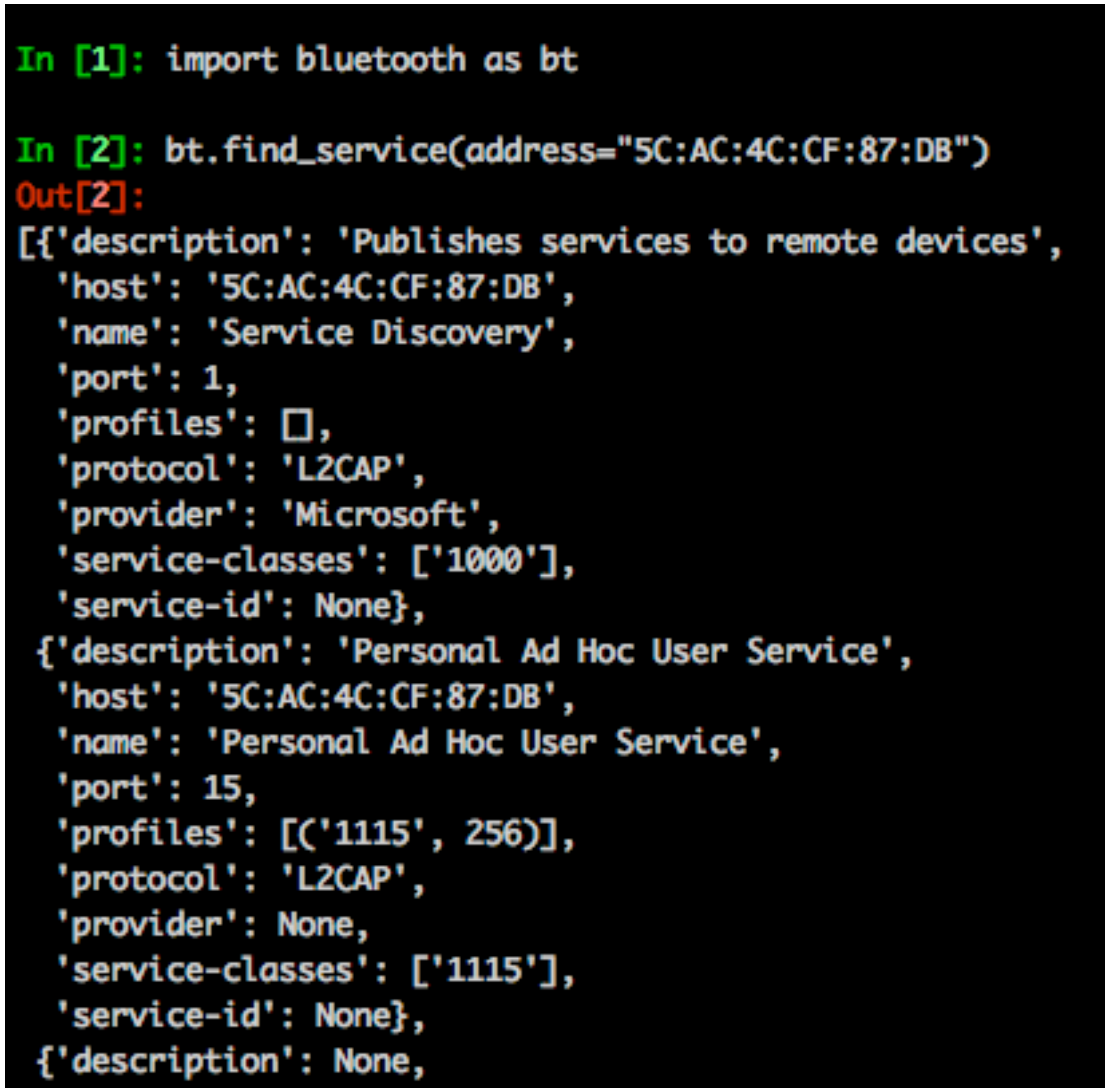
- Does not require discovery mode
- Requires the bluetooth address of the device being scanned
- Returns the device name



```
In [1]: import bluetooth as bt

In [2]: bt.lookup_name("5C:AC:4C:CF:87:DB")
Out[2]: 'T410'
```

Service Inquiry

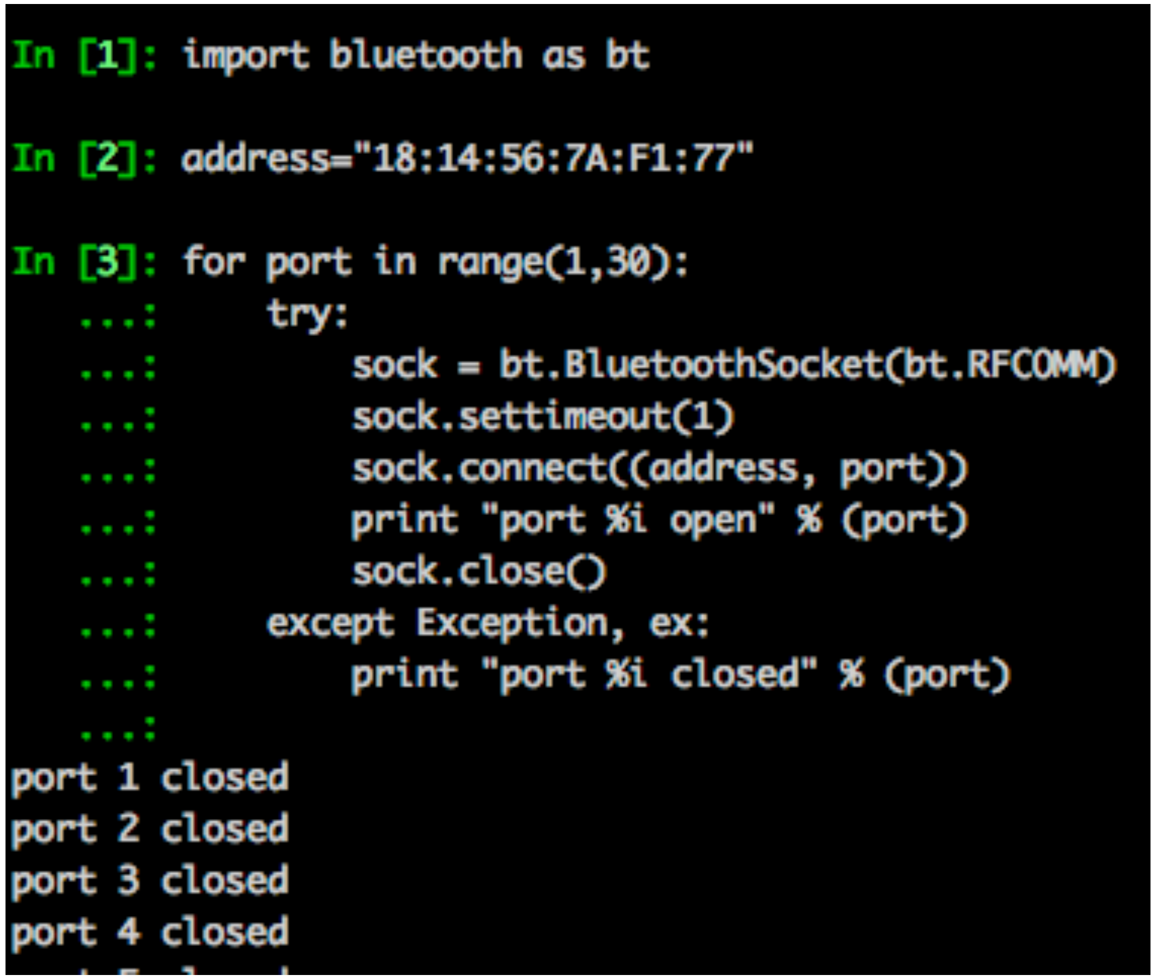
- Requires that the device is in discoverable mode
- Not an instant query
- Returns a list of advertised services
- Requires real NAP



```
In [1]: import bluetooth as bt

In [2]: bt.find_service(address="5C:AC:4C:CF:87:DB")
Out[2]:
[{'description': 'Publishes services to remote devices',
  'host': '5C:AC:4C:CF:87:DB',
  'name': 'Service Discovery',
  'port': 1,
  'profiles': [],
  'protocol': 'L2CAP',
  'provider': 'Microsoft',
  'service-classes': ['1000'],
  'service-id': None},
 {'description': 'Personal Ad Hoc User Service',
  'host': '5C:AC:4C:CF:87:DB',
  'name': 'Personal Ad Hoc User Service',
  'port': 15,
  'profiles': [('1115', 256)],
  'protocol': 'L2CAP',
  'provider': None,
  'service-classes': ['1115'],
  'service-id': None},
 {'description': None,
```

Service Enumeration

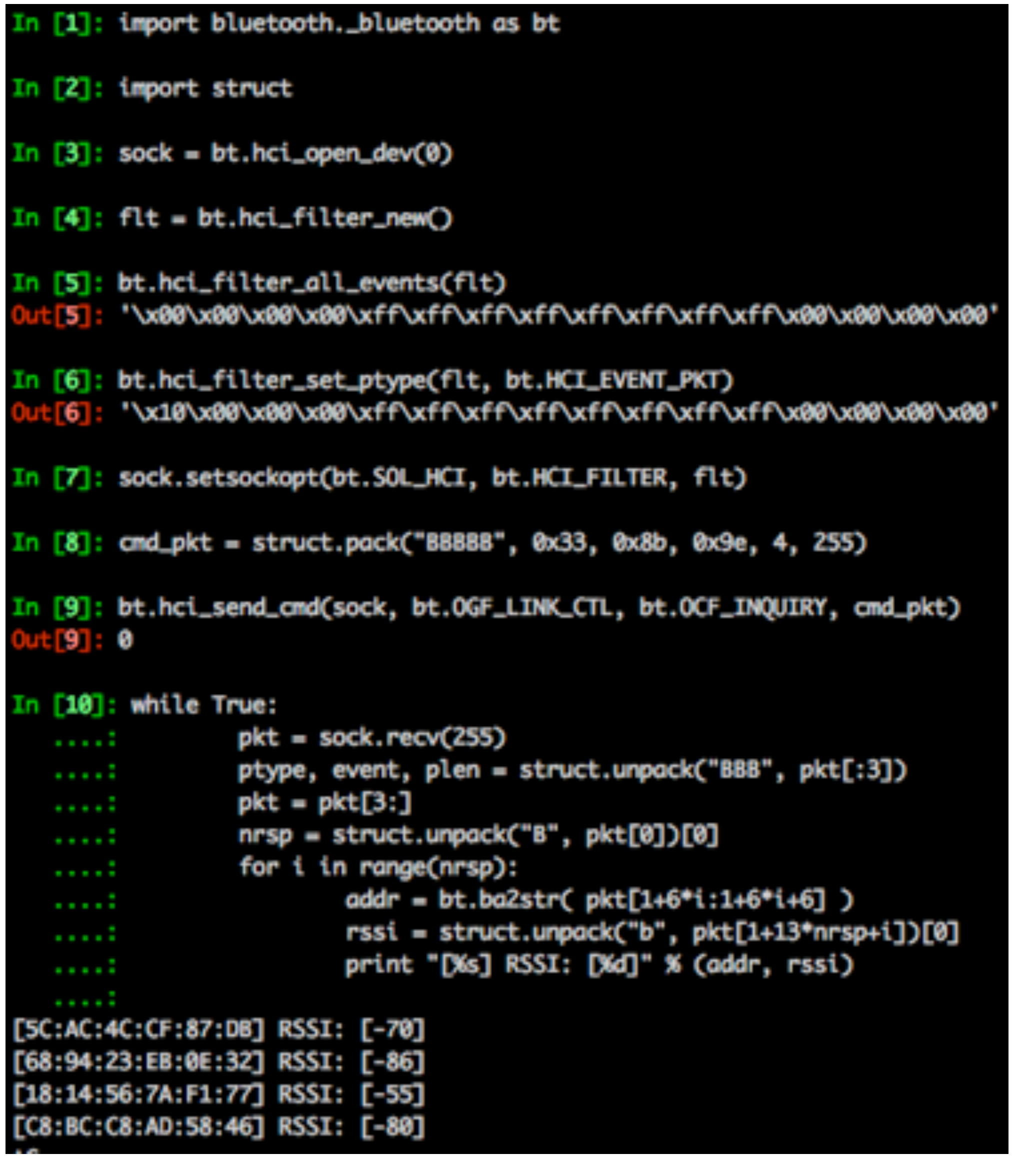
- Does not require discoverable mode
- Requires the target device’s BT address
- Attempts to connect to BT open ports
- Takes a long time to complete for full enumeration



```
In [1]: import bluetooth as bt

In [2]: address="18:14:56:7A:F1:77"

In [3]: for port in range(1,30):
   ...:     try:
   ...:         sock = bt.BluetoothSocket(bt.RFCOMM)
   ...:         sock.settimeout(1)
   ...:         sock.connect((address, port))
   ...:         print "port %i open" % (port)
   ...:         sock.close()
   ...:     except Exception, ex:
   ...:         print "port %i closed" % (port)
   ...:
port 1 closed
port 2 closed
port 3 closed
port 4 closed
```

RSSI Level

- Does not require discoverable mode
- Requires the target BT address
- Queries for the signal strength of the target

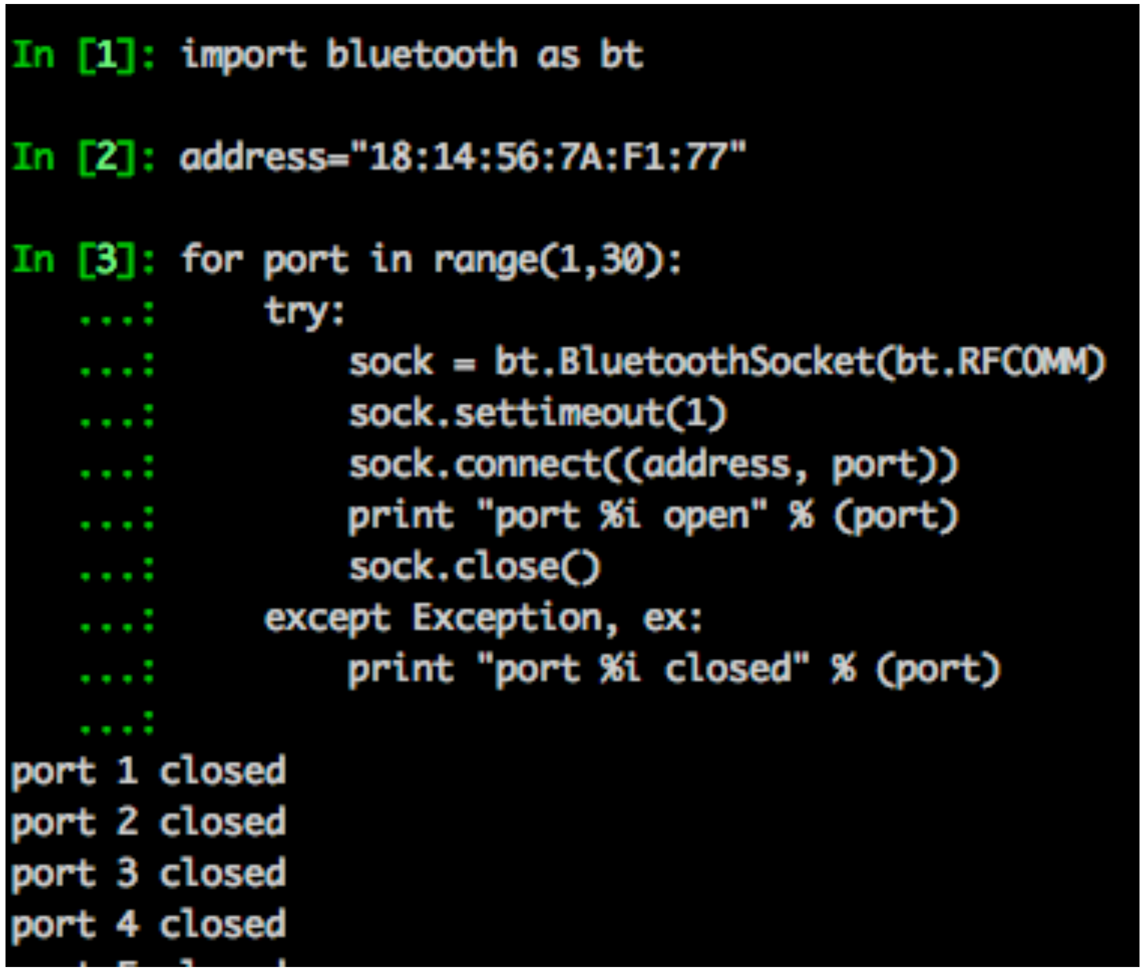


```
In [1]: import bluetooth._bluetooth as bt

In [2]: import struct

In [3]: sock = bt.hci_open_dev(0)

In [4]: flt = bt.hci_filter_new()

In [5]: bt.hci_filter_all_events(flt)
Out[5]: '\x00\x00\x00\x00\xff\xff\xff\xff\xff\xff\xff\xff\x00\x00\x00\x00'

In [6]: bt.hci_filter_set_ptype(flt, bt.HCI_EVENT_PKT)
Out[6]: '\x10\x00\x00\x00\xff\xff\xff\xff\xff\xff\xff\xff\x00\x00\x00\x00'

In [7]: sock.setsockopt(bt.SOL_HCI, bt.HCI_FILTER, flt)

In [8]: cmd_pkt = struct.pack("BBBBB", 0x33, 0x8b, 0x9e, 4, 255)

In [9]: bt.hci_send_cmd(sock, bt.OGF_LINK_CTL, bt.OCF_INQUIRY, cmd_pkt)
Out[9]: 0

In [10]: while True:
   ....:         pkt = sock.recv(255)
   ....:         ptype, event, plen = struct.unpack("BBB", pkt[:3])
   ....:         pkt = pkt[3:]
   ....:         nrsp = struct.unpack("B", pkt[0])[0]
   ....:         for i in range(nrsp):
   ....:                 addr = bt.ba2str( pkt[1+6*i:1+6*i+6] )
   ....:                 rssi = struct.unpack("b", pkt[1+13*nrsp+i])[0]
   ....:                 print "[%s] RSSI: [%d]" % (addr, rssi)
   ....:
[5C:AC:4C:CF:87:DB] RSSI: [-70]
[68:94:23:EB:0E:32] RSSI: [-86]
[18:14:56:7A:F1:77] RSSI: [-55]
[C8:BC:C8:AD:58:46] RSSI: [-80]
```

Authentication

- Can typically be determined by the results of service enumeration

- If ports are deemed open, then auth most likely does not exist



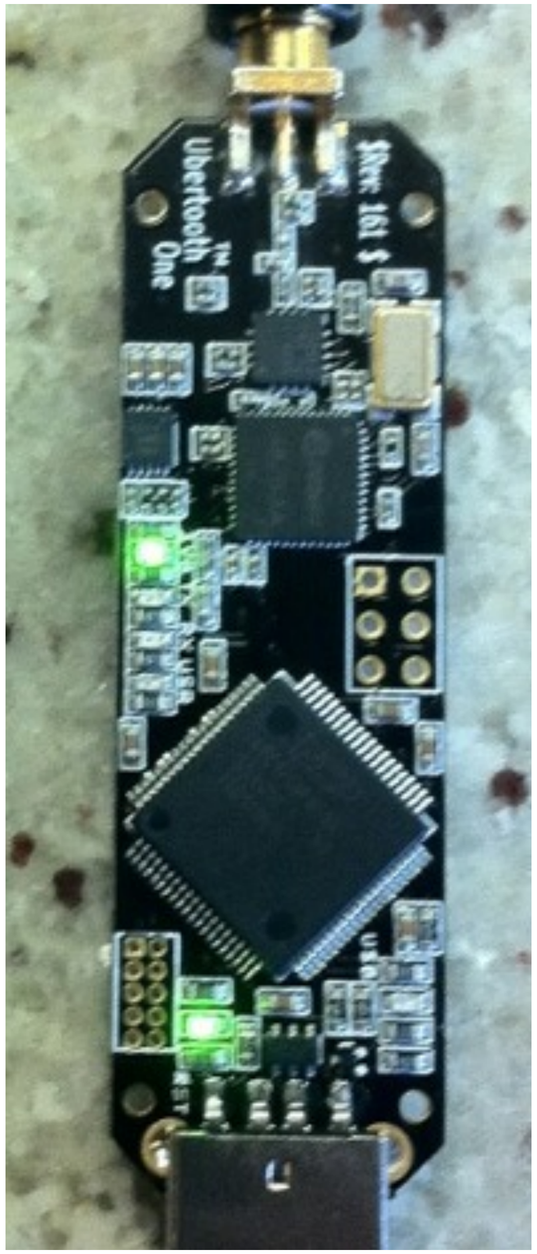
```
In [1]: import bluetooth as bt

In [2]: address="18:14:56:7A:F1:77"

In [3]: for port in range(1,30):
   ...:     try:
   ...:         sock = bt.BluetoothSocket(bt.RFCOMM)
   ...:         sock.settimeout(1)
   ...:         sock.connect((address, port))
   ...:         print "port %i open" % (port)
   ...:         sock.close()
   ...:     except Exception, ex:
   ...:         print "port %i closed" % (port)
   ...:
port 1 closed
port 2 closed
port 3 closed
port 4 closed
```

Passive Scan Techniques

- Requires special hardware
  - Ubertooth, SDR
- Inspects the bluetooth baseband layer

Ubertooth

- Mike Ossman & Dominic Spill
- Provides native tools
- Kismet & Wireshark plugins

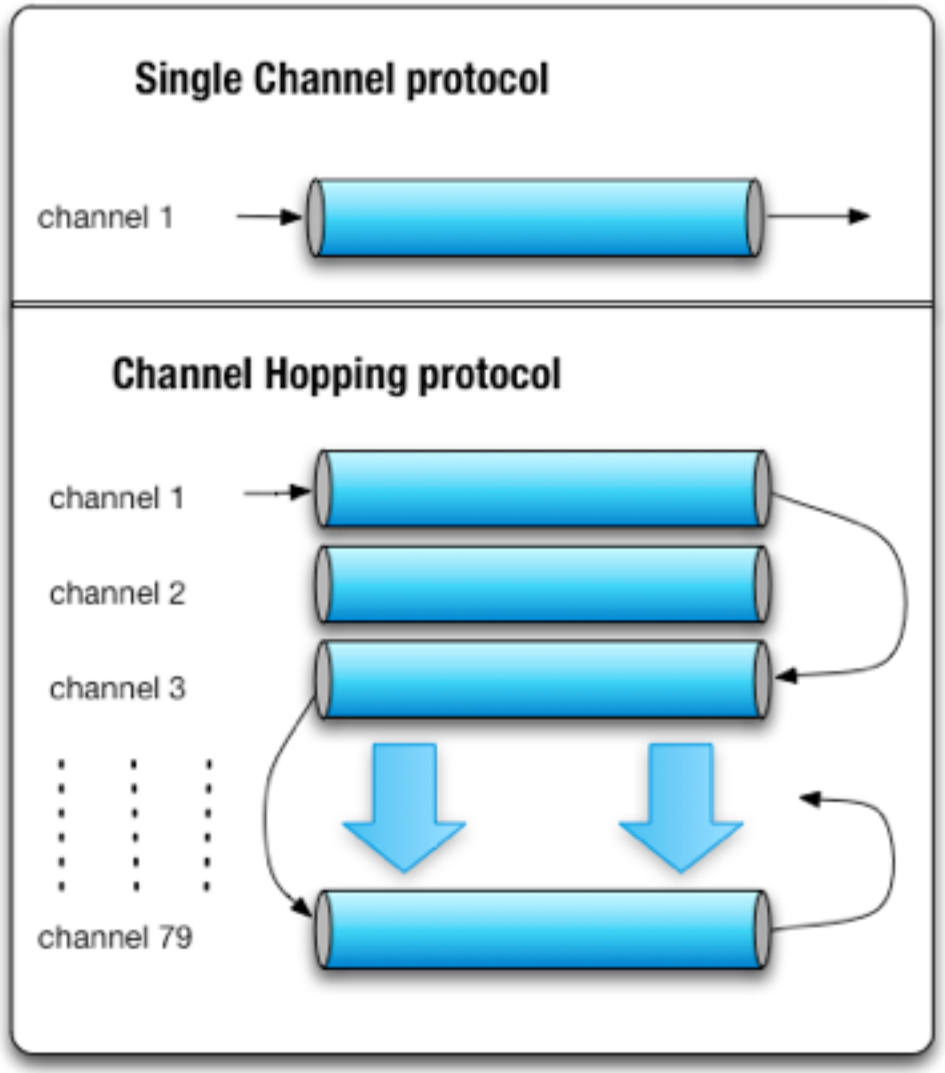


SDR

- USRP
- HackRF
- gr-bluetooth

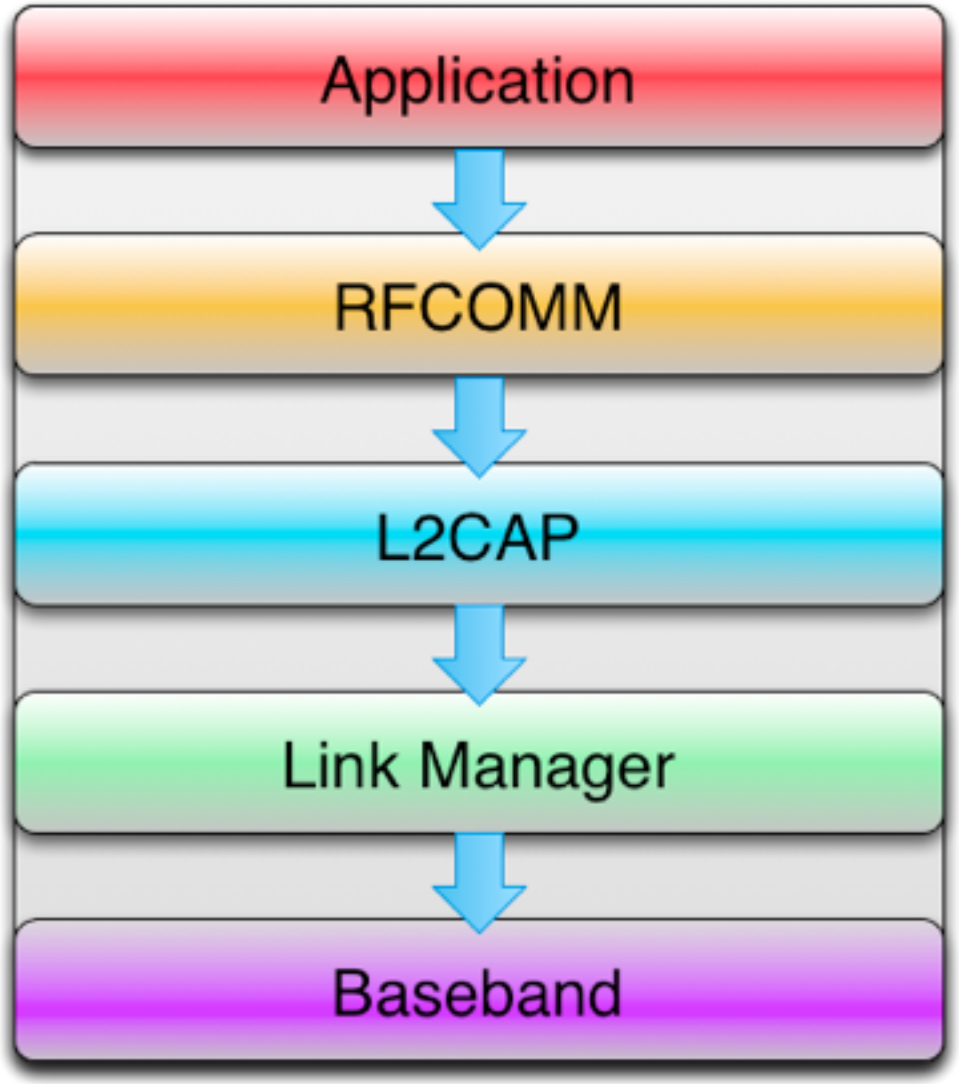
Bluetooth FHSS

- Bluetooth is a frequency hopping protocol



Bluetooth Stack

- BTBB: bluetooth baseband
- BTBB is the air traffic between master and slave BT devices
- passive monitoring happens at the BTBB layer


Application

RFCOMM

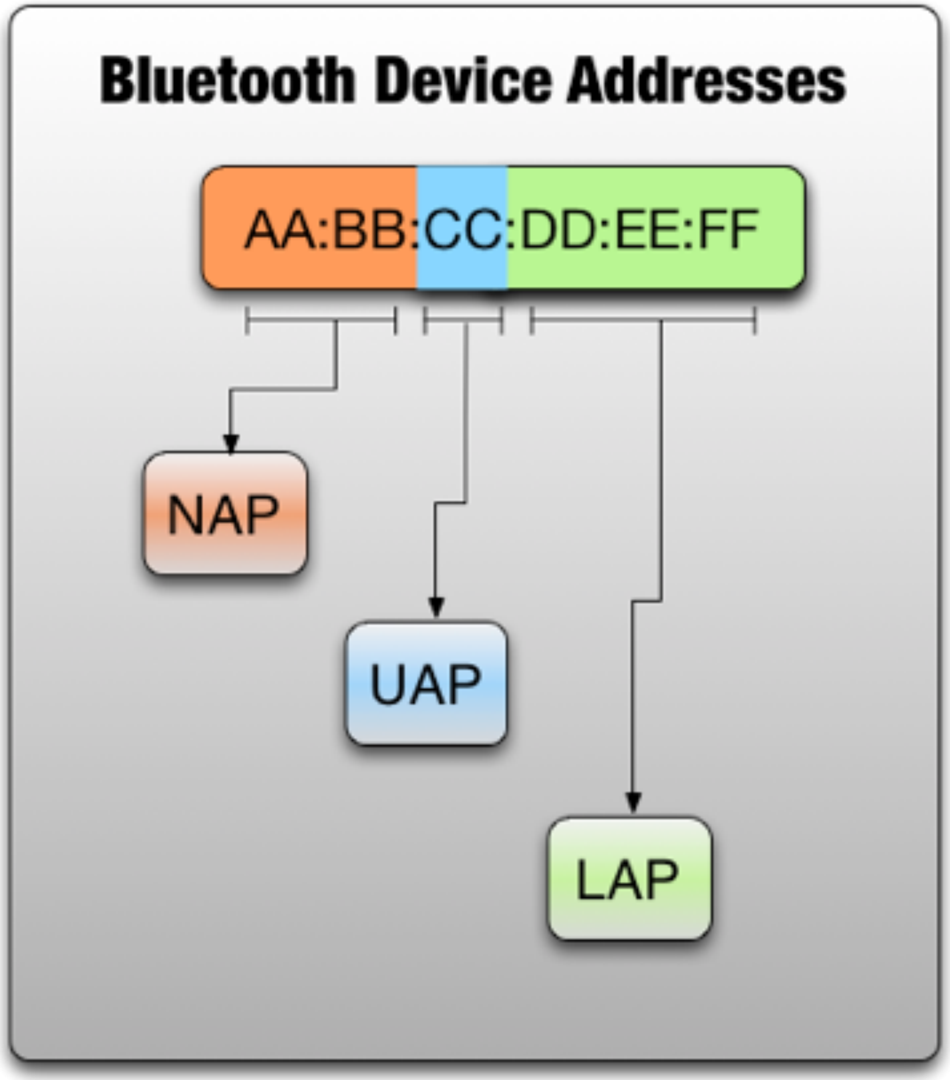
L2CAP

Link Manager

Baseband

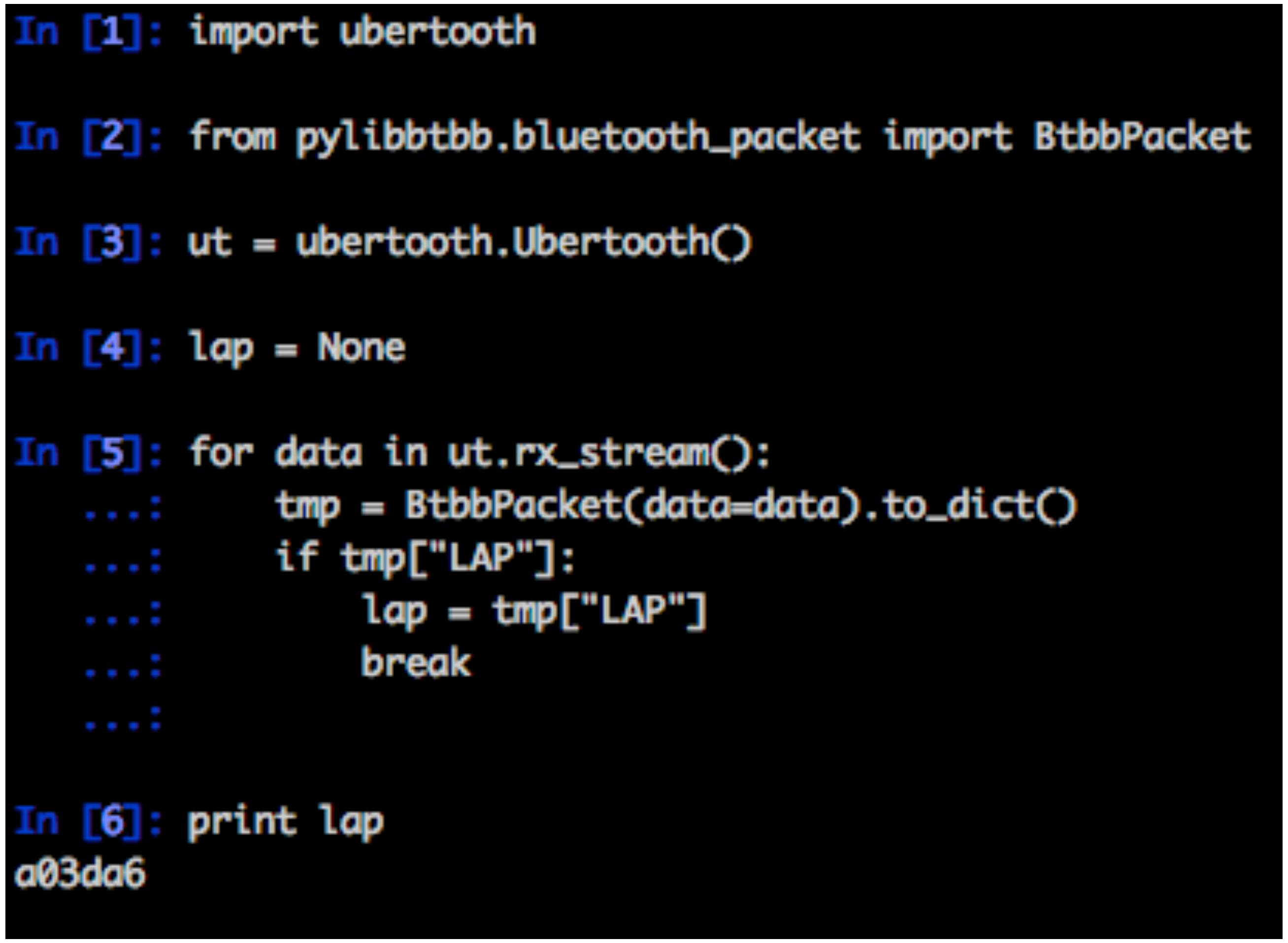
Bluetooth Address

- NAP
  - Non significant
- UAP
  - Upper address
- LAP
  - Lower address



LAP Discovery

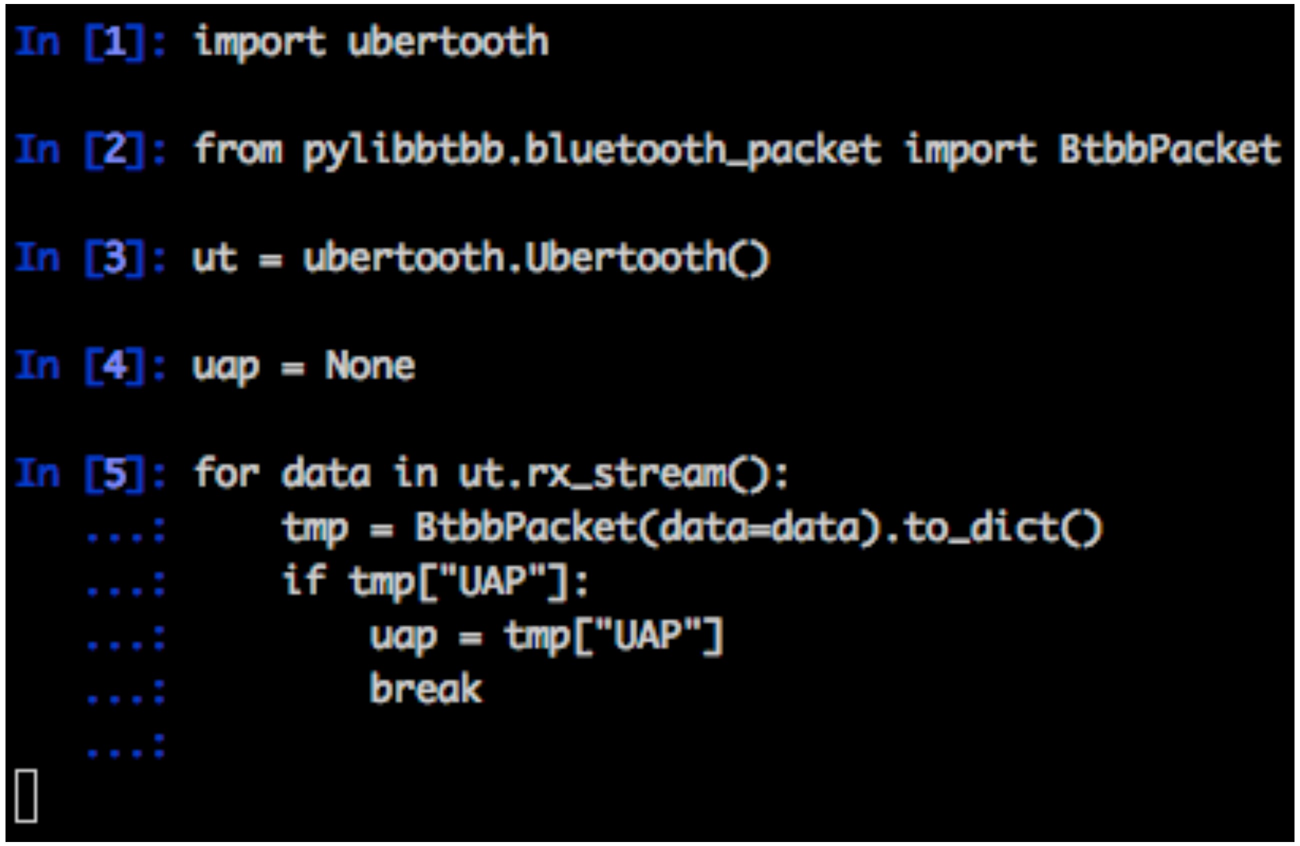
- Possible though
  - Ubertooth tools
  - PyUbertooth
- Passively obtains LAP addresses of nearby devices



```
In [1]: import ubertooth

In [2]: from pylibbtbb.bluetooth_packet import BtbbPacket

In [3]: ut = ubertooth.Ubertooth()

In [4]: lap = None

In [5]: for data in ut.rx_stream():
   ...:     tmp = BtbbPacket(data=data).to_dict()
   ...:     if tmp["LAP"]:
   ...:         lap = tmp["LAP"]
   ...:         break
   ...:

In [6]: print lap
a03da6
```

UAP Discovery

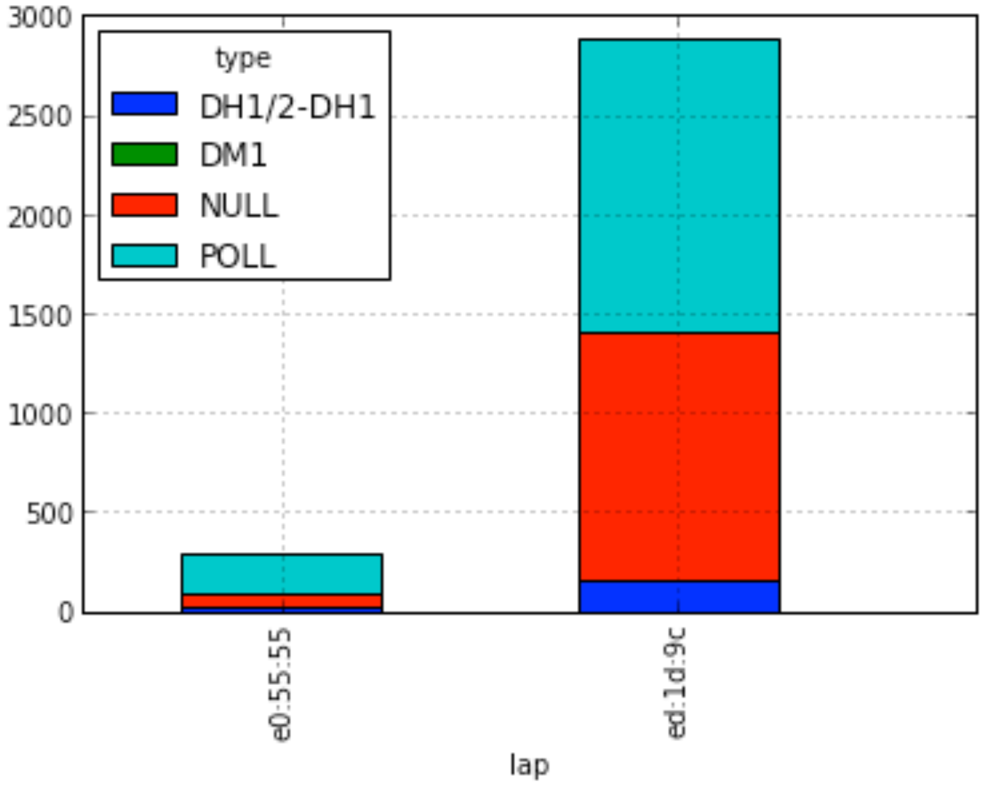
- Can be done though libubertooth
- Can also be done with btbb-scapy
- Yet to be implemented in pyUbertooth
- Can also be implemented in python with bash wrappers



```
In [1]: import ubertooth

In [2]: from pylibbtbb.bluetooth_packet import BtbbPacket

In [3]: ut = ubertooth.Ubertooth()

In [4]: uap = None

In [5]: for data in ut.rx_stream():
   ...:     tmp = BtbbPacket(data=data).to_dict()
   ...:     if tmp["UAP"]:
   ...:         uap = tmp["UAP"]
   ...:         break
   ...:
```

Packet Types

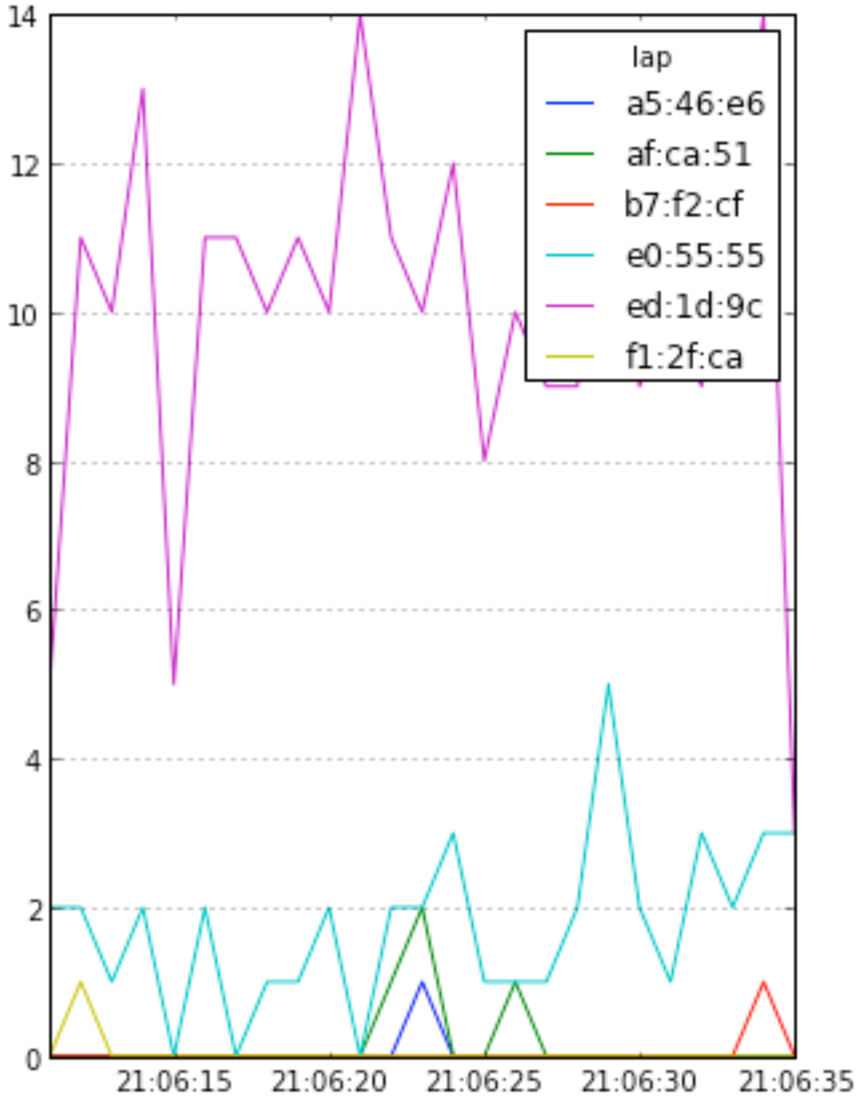
- Currently obtainable though Ubertooth pcap dumps and python analysis
- Rudimentary support in pyubertooth



Packet Volume

- Currently obtainable though Ubertooth pcap dumps and python analysis

- Rudimentary support in pyubertooth



Misc Techniques

- Use probability
- Use other non-bluetooth technologies

Vendor Matching

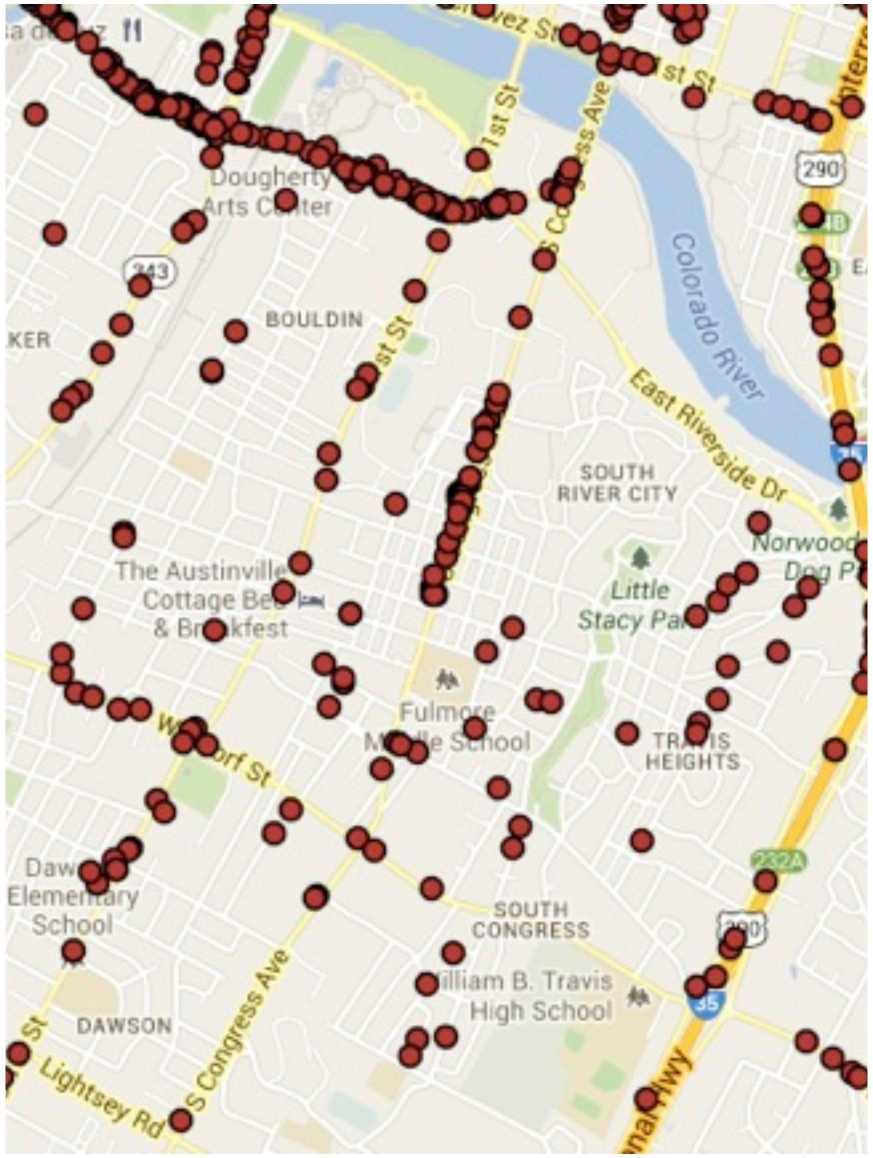
- Match NAP+UAP to known vendor lists
- Wireshark manuf file
- Public OUI vendor list
- Rudimentary support in scapy-btbb

Service to OS Matching

- Currently there is no large scale bluetooth service enumeration list
- Vendor and NAP could be determined by service matching

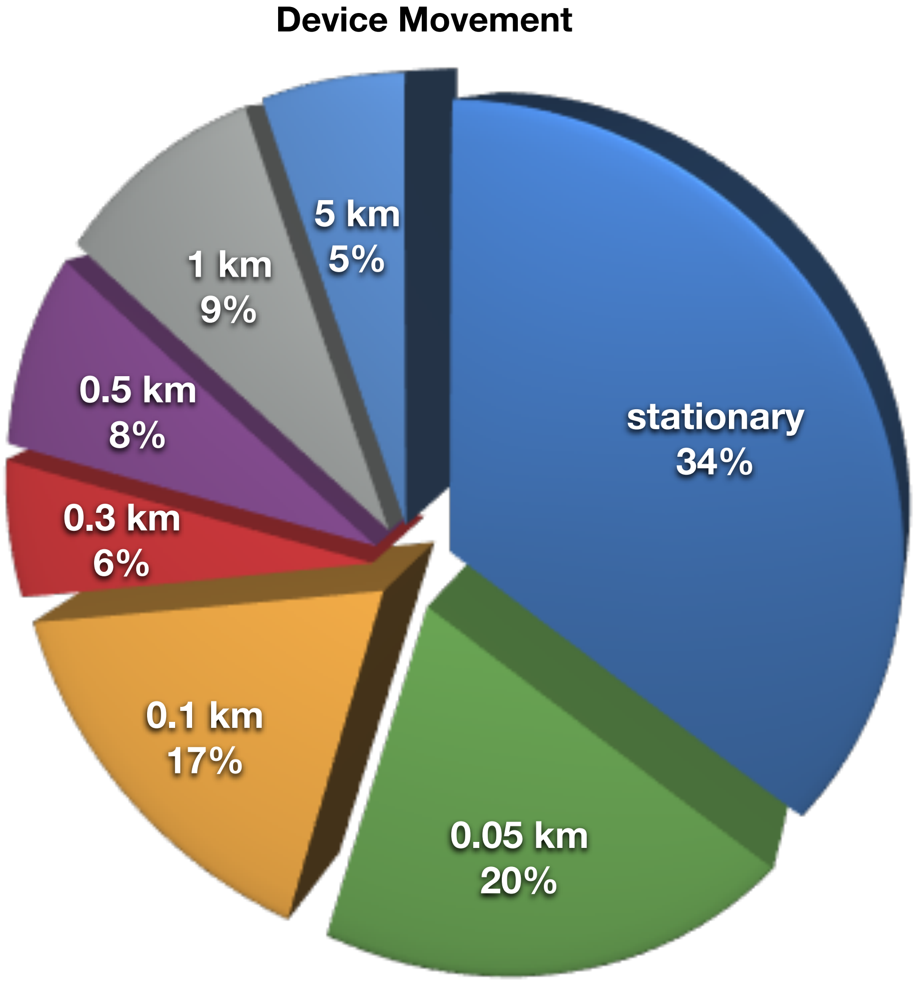
Geolocation Tagging

- Geolocation services depend on the base OS
- Corrilate a geolocation to each device sighting
- Done in the Bluetooth Database Project



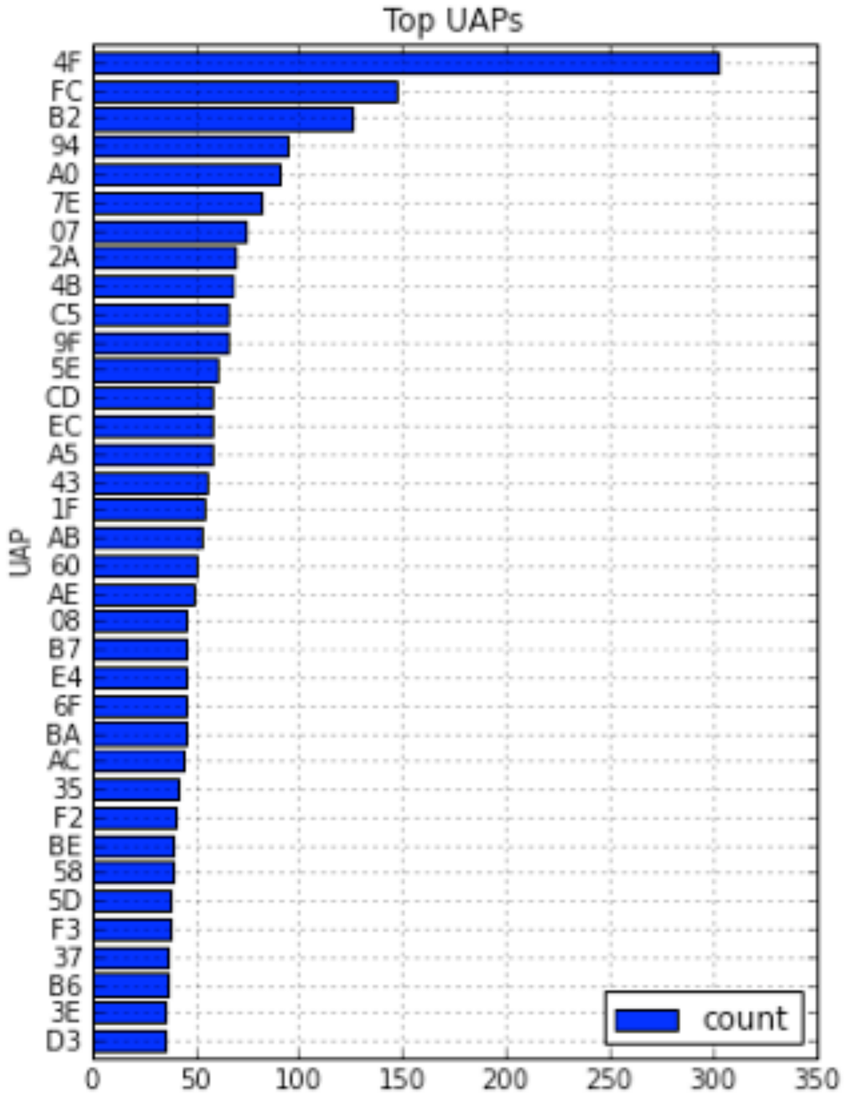
Movement

- Static vs moving sensor
- Multiple sightings required
  - 1,700 unique devices with 2+ sightings
- Computed by a devices 2 farthest points



UAP Probability

- 246 of 256 possible UAP's used in large surveys
- Some UAPs are more probabile
- Can be tested via bruteforce and probability



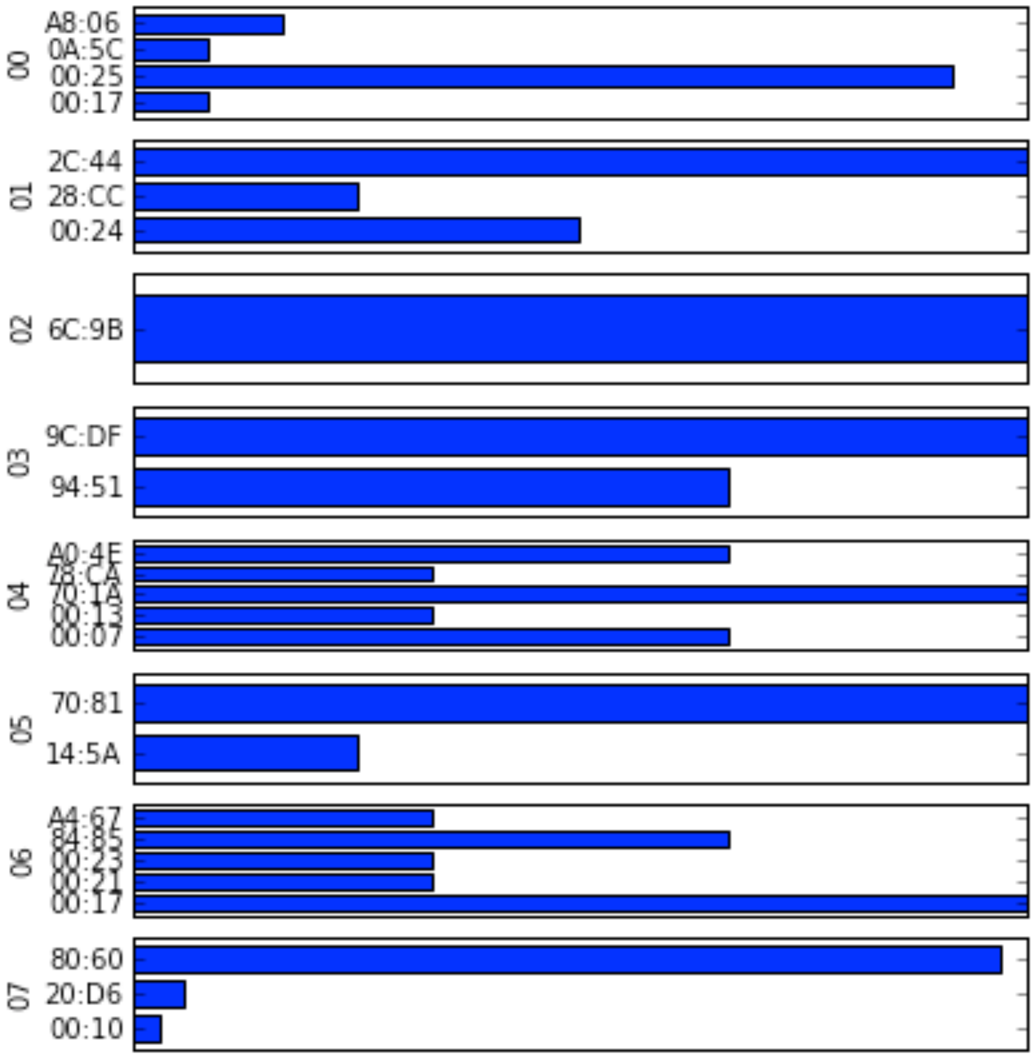
NAP Probability

- Only a small amount of the address space is used 554 / 65,536

- Predict NAP based on UAP

- Majority of UAP instances have only 1-3 associated NAP

- Worst case is 8 NAPs



Direction by Doppler

- Balint Seeber
- SDRDF
    - Software Defined Radio Direction Finding
- Requires antenna Array
- Could possibly be done with Uberteeth

Hybrid Techniques

- Mixing of
  - Active techniques
  - Passive techniques
  - Misc techniqes

Active Mix

- Discoverable
- Service enumeration
- RSSI queries

Passive LAP + Active

- Passive
  - LAP discovery
- Active
  - UAP brute force

Passive UAP + Active

- Passive
  - Find LAP & UAP
- Active
  - Name lookup
  - Service enumeration

Passive/Active/Prob

- Passive
  - LAP & UAP
- Active
  - Name query
- Probability
  - NAP probability by UAP

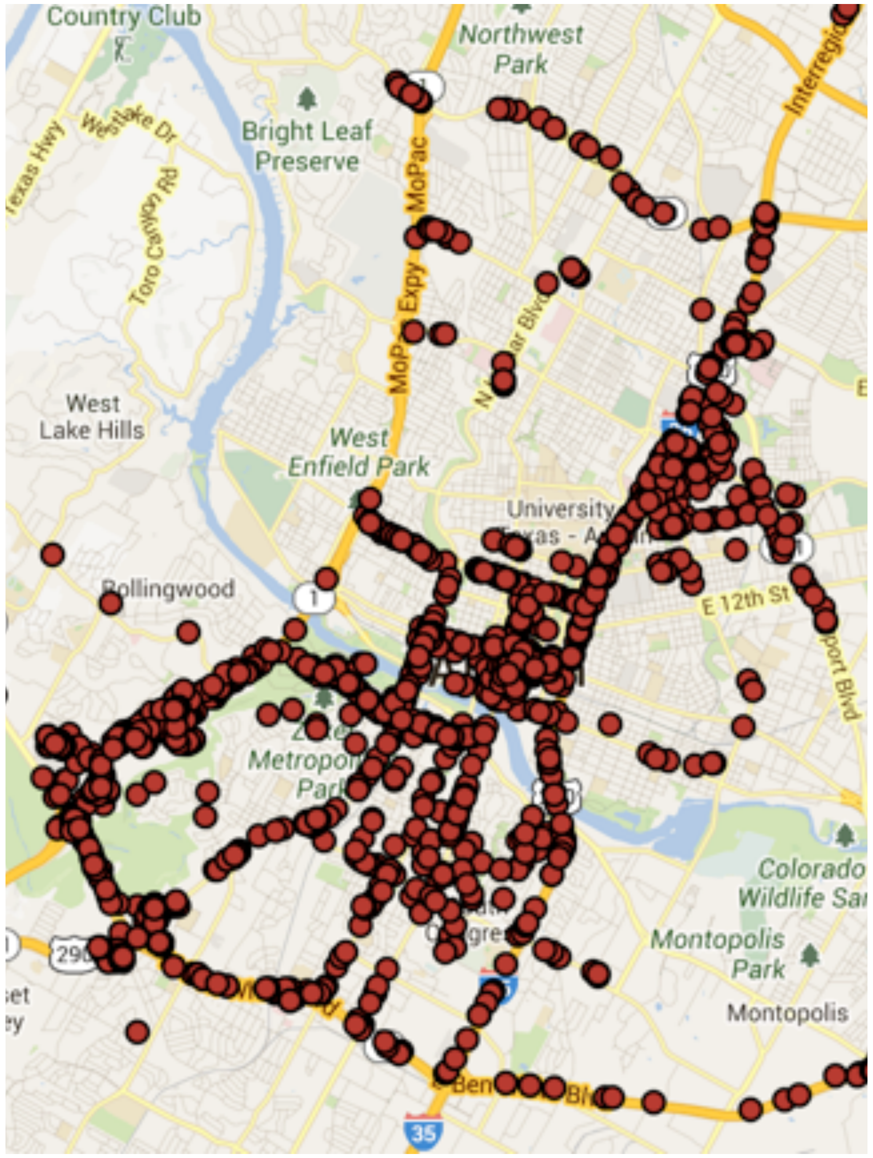
Passive/Active/Lookup

- Passive
  - LAP & UAP
- Active
  - Service Enumeration
- Probability
  - NAP probability by service

So what can you do with bluetooth scanning?

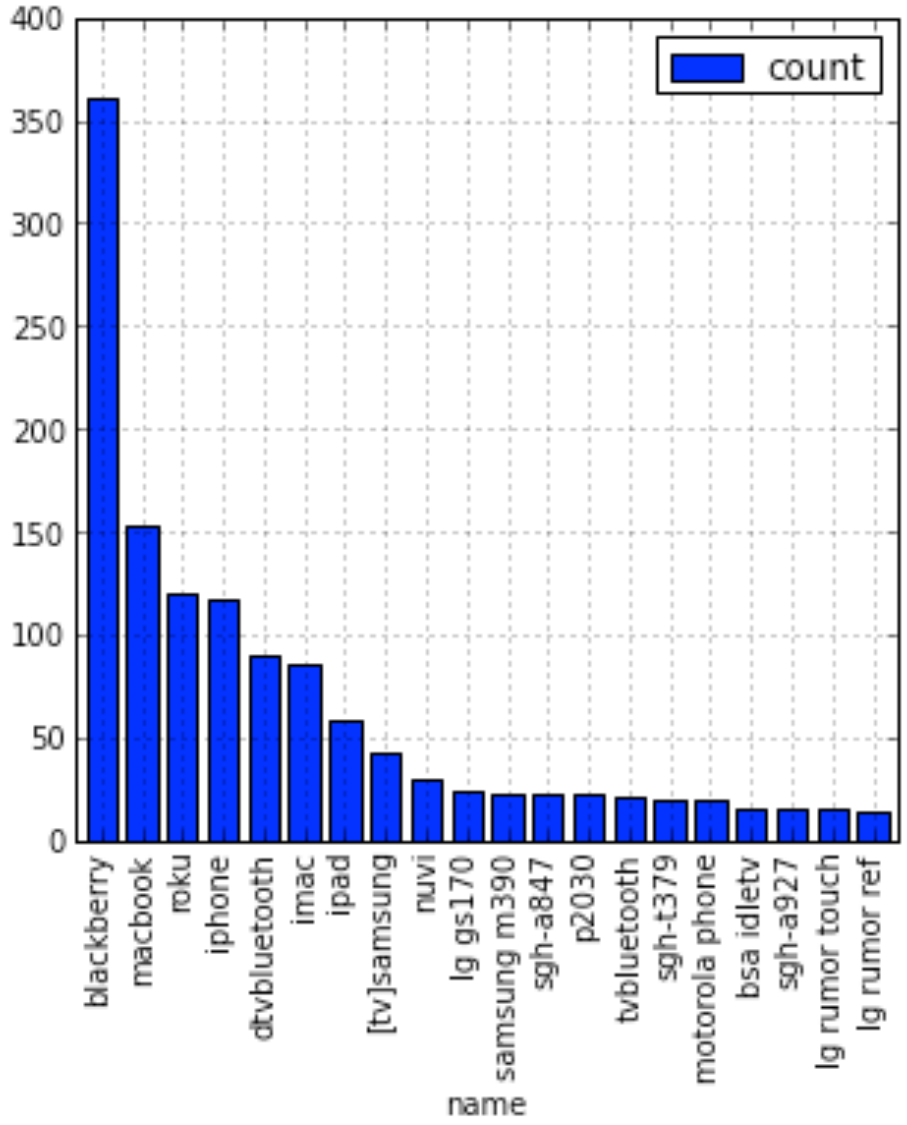
BluetoothDatabase

- Defcon 2013
- Active scanning & Geolocation
- bluetoothdatabase.com
- ~5k unique devices
- +12K device sightings



BT Vulnerabilities

- Active bluetooth vulnerabilities do exists



How can I do all of this stuff?

Ubertooth

- Provides a core set of tools
  - ubertooth-rx
    - uap, lap, clock passive discovery
- BTLE
- Pcap capture

PyUbertooth

- Pure python
- Direct Ubertooth interface via python
- LAP discovery
- Possible UAP, packet type and traffic volume analysis

pyBluez

- Python interface to Bluez
- Support on Linux
- Allows for low level Bluez functionality

Blucat

- Java based bluetooth scanner
- Provides discoverable, service, etc scanning
- Also provide easy mechanisms for:
  - piping data though bluetooth
  - bluetooth service testing

BlueScan

- New python BT project
- Utilizes Active and Passive techniques
- Incorporates BT APIs, Ubertooth and other mechanisms
- Alpha release soon...
- Stable release in 2014