556.3

Exploiting Wireless IoT: Wi-Fi, BLE, Zigbee, LoRa, and SDR

SEC556.3 | IoT Penetration Testing

SANS

Exploiting Wireless IoT: Wi-Fi, BLE, Zigbee, LoRa, and SDR

Welcome to SANS Security SEC556, *IoT Penetration Testing*. In this course, we'll take a deep look at the threats as well as the attack and defense opportunities surrounding wireless networks. Our primary focus in the course will be on evaluating IoT devices but we'll take a more holistic approach and examine much of the IoT ecosystem, including the hardware, networks, popular wireless protocols and messaging services.

Let's keep this session interactive. If you have a question, please let the instructor know. Discussions about relevant topics are incredibly important in a class like this, as we have numerous attendees with various levels of skill coming into the class. Share your insights and ask questions. The instructor does reserve the right, however, to take a conversation offline during a break or outside of class in the interest of time and the applicability of the topic.

As the course author, we welcome any comments, questions, or suggestions pertaining to the course material:

Larry Pesce
larry.pesce.556@gmail..com
Twitter: @haxorthematrix

James Leyte-Vidal
jameslvsec556@gmail.com
Twitter: @jamesleytevidal

Table of Contents

Each course book includes a table of contents for each of the major topics and exercises that we'll cover.

Course Roadmap

Introduction to IoT Network Traffic and Web Services

Exploiting IoT Hardware Interfaces and Analyzing Firmware

Exploiting Wireless IoT: Wi-Fi, BLE, Zigbee, LoRa, and SDR

SECTION 3

1. **Wi-Fi**
   Exercise: Wi-Fi PSK Cracking
2. **Bluetooth Low Energy**
   Exercise: BLE Device Interaction
3. **Zigbee**
   Exercise: Zigbee Traffic Capture
4. **LoRa**
5. **SDR**
   Exercise: Conducting a Replay Attack on IoT

Course Roadmap

In addition to the table of contents, each course book includes a course roadmap to show our progress as we work through each topic and its exercises. The current topic will be underlined. Previous topics will be shown in a light gray typeface to indicate completion.

Introduction

Wi-Fi Origins
Wi-Fi at the ones and twos
WEP
WPA
WPA2
WPA3

Introduction

In this module, we will examine Wi-Fi.

We will start with a bit of history on Wi-Fi, and the regulatory organizations that govern it.

From there, we'll move into capturing Wi-Fi traffic, the various modes of operation for a Wi-Fi adapter, and some detail on the 802.11 frame header.

We will detail all the various methods of wireless encryption available, and common attacks against them.

Finally, we will discuss how to operationalize this all.

Why Wi(-Fi)?

Ubiquitous technology stack

Available in a large number (read: most) IoT devices

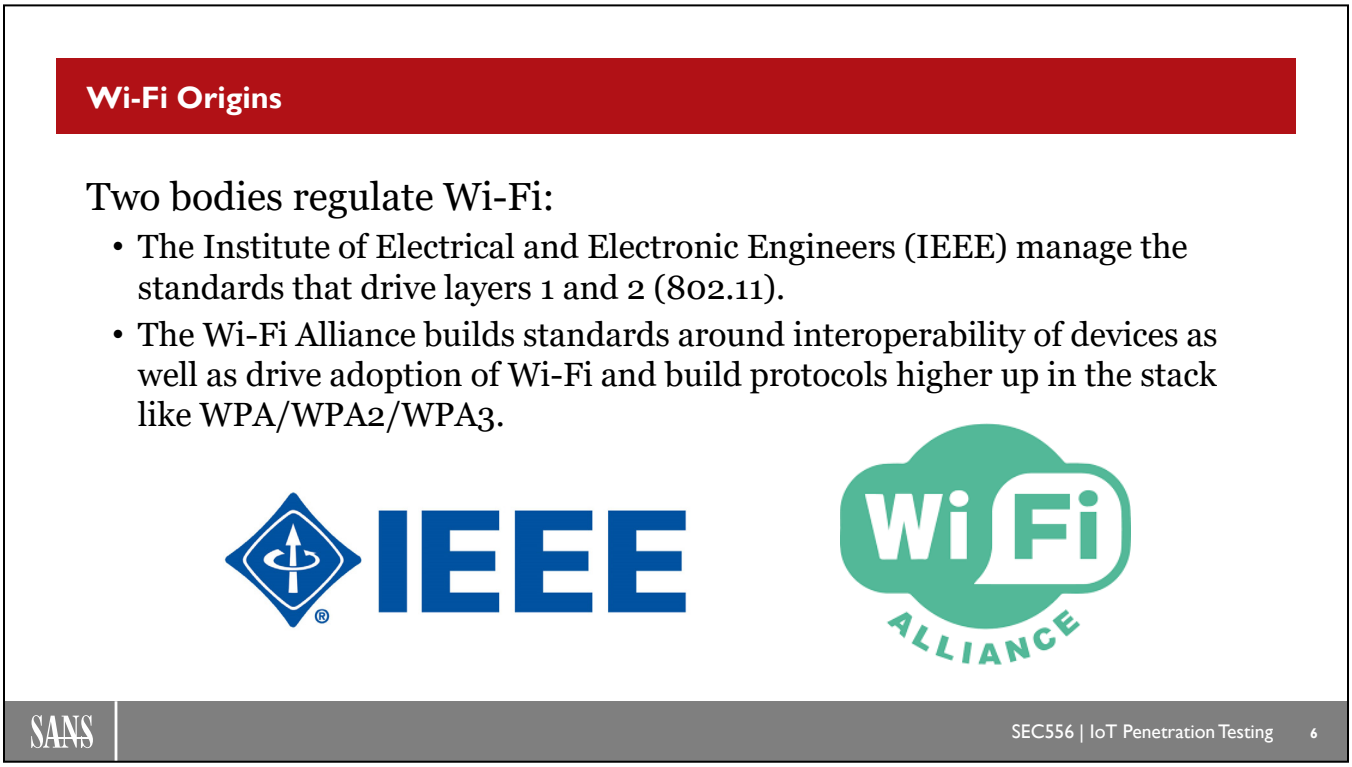
While newer, low power technologies may be favored in some implementations, they're still very commonly available (often alongside those other protocols).

Why Wi(-Fi)?

Wi-Fi is probably the most common protocol you will see on IoT devices today. However, it has somewhat fallen out of favor for day-to-day use for smaller, embedded devices, due to the overall high-power requirements of Wi-Fi.

However, none of these lower power, lower data rate protocols can beat Wi-Fi for raw throughput speed, and so for certain applications, Wi-Fi is still the clear winner.

Unfortunately, Wi-Fi has had some security missteps over the years, and so the relative ubiquity of this protocol makes it a common (and easily interacted with) attack surface.



Wi-Fi Origins

While protocols like Bluetooth have but one guiding body, Wi-Fi has two: The Wi-Fi alliance and the Institute of Electrical and Electronic Engineers, or IEEE.

The IEEE deals with a lot of the layer 1 and 2 technology in Wi-Fi – how signals are placed in the air, modulation/demodulation, etc.. Additionally, the IEEE deals with the technological advancements that allow for greater throughput of Wi-Fi – so when you think of technologies like Multiple Input/Multiple Output (MIMO), beamforming, and the like, these are coming to the standards from the IEEE. When you think about the standards like 802.11b, 802.11g, 802.11n, 802.11ac, and 802.11ax (and others), these are IEEE standards.

The IEEE makes their standards available for public review, though newer standards such as the 802.11-2020 standard require access either via payment, or an IEEE subscription. However, older versions of documents, such as the superseded 802.11-2016 revision, may be available for electronic download with only a registration.

On the other hand, the Wi-Fi Alliance originally grew out of a desire to make various Wi-Fi hardware interoperable. While it might be surprising in this day, manufacturer 'lock-in' in the earlier days of Wi-Fi was not unheard of. The Wi-Fi Alliance developed the Wi-Fi certification programs and certification marks, which assured the user that any device with the Wi-Fi logo would work with any other device with the same logo. Additionally, the Wi-Fi Alliance develops and supports standards higher up in the stack, such as those associated with wireless encryption. WPA, WPA2, and WPA3 are products of the Wi-Fi Alliance. The Wi-Fi Alliance has also been responsible for certain branding efforts associated with Wi-Fi at large – for example, the IEEE 'alphabet soup' of numerous 802.11 standards was confusing to the users, and so in 2018, the Wi-Fi Alliance 'rebranded' these standards to an easier to understand numbering scheme:

802.11 – Wi-Fi 1
802.11b – Wi-Fi 2
802.11a/g – Wi-Fi 3
802.11n – Wi-Fi 4
802.11ac – Wi-Fi5
802.11ax – Wi-Fi 6

The IEEE can be accessed online at https://www.ieee.org/ , and the Wi-Fi Alliance at https://www.wi-fi.org/.

Wi-Fi at The Ones and Twos

802.11 layers another header to a packet.

- This 802.11 frame header handles operations like advertising networks, connect/disconnect from networks, and sending data.

This header is normally stripped out upon receipt of the packet for 'typical' wireless cards and drivers.

- At this point, a wireless frame looks like a normal Ethernet frame.

This complicates things for penetration testers.

- We can't review frames for other wireless networks; we need to be associated (join) to a particular network to capture/interact with traffic.
- If we can see the headers, we could interact with other networks traffic – we can't segment the air, can we?

Wi-Fi at the Ones and Twos

Wi-Fi traffic adds another layer to a packet – and when we say layers, think about the layers you observe in Wireshark that (mostly) map to the OSI model. Because Wi-Fi adds the additional complexity of channels, radios, and interference to the communication, another frame header for 802.11 is added into the mix.

When using a traditional wireless interface, as you may be using now, this is mostly transparent to the user as the 802.11 headers, once they've served their purpose by getting the signal to its destination wired network or device, strip the 802.11 headers away.

However, as penetration testers, the information transmitted in the 802.11 headers is enormously important! It can help us identify nearby wireless networks, their capabilities, and their weaknesses. In many cases this data can allow us to access an encrypted wireless network without even having to send a packet. We need to be able to see this 802.11 data for maximum success in our engagements.

Wi-Fi Adapter Modes

802.11 supports 4 modes of operation for your Wi-Fi 'card' (adapter):

- Master
- Managed
- Ad-hoc
- Monitor

Wi-Fi Adapter Modes

Wi-Fi adapters can operate on one of four modes, only one at any given time.

Master mode means that this particular wireless adapter is acting as an access point, managing a network, and allowing others to connect, negotiate capabilities, and send traffic. While we normally think of a full-size access point in this role, many Wi-Fi adapters can act as an access point – in fact, the Wi-Fi card in your hardware kit permits this.

Managed mode is the typical mode of operation most people think of when they think of Wi-Fi from a client perspective. Wi-Fi adapters in managed mode can associate and authenticate to an access point and send data.

Ad-hoc mode is another type of Wi-Fi where there is no concept of a client or an access point. All devices in an ad-hoc network are peers. Ad-hoc networks are less common and used for quick transactions of some kind of data. For example, it is not uncommon to observe a wi-fi network out in the wild called hpsetup. This is an ad hoc network used by HP print devices to allow for simple connections from clients to send a print job with minimal complexity.

Finally, monitor mode allows for passive observation of data. An adapter in monitor mode cannot be joined to a network, but it can observe any traffic being sent in the frequencies the adapter supports.

Wi-Fi Adapter Modes Cont'd.

You will commonly interact with adapters in Managed mode.

- Devices connecting to a network, with an AP running its adapter in Master mode

In this configuration, we can capture traffic for that network.

Wi-Fi Adapter Modes Cont'd.

Based on your previous slide, you may infer that only Monitor mode is useful for packet capture and research, but that is not completely true. Managed mode adapters can also perform packet captures and help us get an initial 'lay of the land', with one critical limitation. We have to associate to a network, and we will only see the traffic for that network.

Managed Mode Packet Capture

Unfortunately, in Managed mode, we don't get to see any 802.11 data.

We also have to be joined (associated) to that network.

There are three types of 802.11 frames; we can only see one (data).

BUT we can see any data for that network within range of our radio (remember hubs?).

```
$ sudo tcpdump -neti wlan0
0:4:23:63:88:d7 0:c:41:3f:31:3c 0800 120: 172.16.0.100.4213 > 65.173.218.103.5223: P
1540033229:1540033295(66) ack 56756900 win 17520 (DF)
0:c:41:3f:31:3c 0:4:23:63:88:d7 0800 54: 65.173.218.103.5223 > 172.16.0.100.4213: . ack
66 win 11792 (DF)
```


Managed Mode Packet Capture

This means that a managed mode packet capture will not show us data for other networks, even if they are using the same Wi-Fi bandwidth of the network we are associated to. That information is stripped out before we can see it in managed mode. Our data in a packet capture will look similar to an Ethernet packet capture – with one important advantage – we can see all the traffic for all the clients around us.

You might have performed packet captures in the past from a hub, a piece of networking gear that did not differentiate traffic sent down different ports of the device; all ports got all the traffic. In Wi-Fi, it is the same way – we can't segment the bandwidth among the clients, so every client sees traffic for every other client – but in managed mode, only for clients on that particular network.

The above screenshot shows a packet capture using tcpdump on a managed mode interface. The packet capture looks similar to a packet capture you would run against a wired Ethernet network – MAC addresses, IP addresses and ports, and TCP flags.

What about Monitor Mode?

Monitor mode removes many of these constraints

- We can observe *any* signal in radio range (no association to a network).
- We can see networks looking for clients (or clients looking for networks).
- If these frames do not use encryption, they can be casually observed by *anyone* who can put their interface into monitor mode.

Monitor mode implies *passive monitoring* (no packets sent)

- It is possible to inject packets into the air in monitor mode with some cards, with the assistance of modified (hacked) drivers.

Being able to access data in monitor mode is critical for penetration testing.

What about Monitor Mode?

Monitor mode, on the other hand, removes many of these constraints. Without associating to a network, we can observe any traffic within range of our radio. We can also observe stray clients sending probe requests for networks they have connected to previously, as well as networks beaconing their capabilities. Finally, we can observe data that is transmitted unencrypted.

Monitor mode captures are passive – no data is sent while this data is collected. This means an adversary (or us as penetration testers) can observe data with minimal risk of detection. All that being said, a monitor mode interface can transmit data, with modifications to the drivers. This can however cause instability in the network interface.

Monitor Mode Interfaces

Once an interface is in monitor mode, a whole new world opens up to us.

- We get access to all three types of 802.11 frames (management, data, and control).
- We can see other networks, assess their security posture, know what channel they use, and access non-encrypted data *without sending a single packet*.

```
$ sudo tcpdump -neti mon0
BSSID:0:c:41:3f:31:3e DA:ff:ff:ff:ff:ff:ff SA:0:c:41:3f:31:3e Beacon (somethingclever)
[1.0 2.0 5.5 11.0 18.0 24.0 36.0 54.0 Mbit] ESS CH: 11 , PRIVACY
BSSID:0:c:41:3f:31:3e SA:0:4:23:63:88:d7 DA:ff:ff:ff:ff:ff:ff Data IV:14f75 Pad 0 KeyID
0
BSSID:0:c:41:3f:31:3e SA:0:4:23:63:88:d7 DA:0:c:41:3f:31:3c Data IV:14f76 Pad 0 KeyID 0
```


Monitor Mode Interfaces

Running the same command from the previous slide, but against a monitor mode interface, shows wildly different data.

3 packets are shown in this slide. The first packet is a beacon frame, which is a type of 802.11 frame known as a management frame. This are packets you will only see in monitor mode. This beacon frame advertises the capabilities of an access point (data rates, channel, encryption support, etc.).

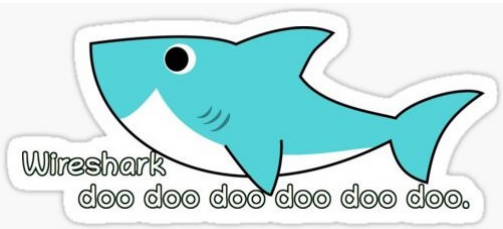
The following two frames are data frames, which would be visible from a managed mode packet capture. However, the IV (initialization vector) data would not be available in managed mode, as this is contained in the 802.11 header. In this case, this is data traffic using WEP to provide data encryption - such as it is. More on that later in this module.

Capturing Traffic in Monitor Mode

Windows
- Libraries (Npcap) and software (Wireshark/Tshark)

macOS
- Native capabilities with airport utility

Linux
- Libraries (libpcap) and software (tcpdump/wireshark/etc.)



AirPort

Capturing Traffic in Monitor Mode

You are hopefully convinced at this point that monitor mode packet captures are important. How do we perform these?

In this class, we will focus on packet captures in Linux. However, we would be remiss not to detail how to perform these in other common operating systems.

In Windows, most wireless adapter drivers since Windows Vista support placing an interface into monitor mode. However, the software to actually perform that action was lacking. More recently, options have become available in this space. Npcap (from the developers of nmap) exposes an API for the capture of packets in monitor mode in Windows. Combined with a packet capture utility like Wireshark or tshark, we have a fully functional packet capture setup in Windows with a few software addons.

Npcap is developed and maintained by the team behind nmap and is available at https://nmap.org/npcap/ . Wireshark (originally Ethereal) is developed by Gerald Coombs and a community of contributors and is available at https://www.wireshark.org/.

In macOS, native monitor mode capture capabilities are available via the airport utility. In some releases such as macOS Mojave, GUI workarounds were required, but these were remedied with the release of macOS Catalina.

Placing an Interface into Monitor Mode - Linux

We can place an interface into Monitor mode using a combination of the iw and ip commands.

That is a lot of typing...

```
# iw dev wlan0 interface add mon0 type monitor
# ip link set mon0 up
# iw dev mon0 set channel 1
# iw dev mon0 info
Interface mon0
        ifindex 6
        wdev 0x200000002
        addr 00:c0:ca:85:00:ba
        type monitor
        wiphy 2
        channel 1 (2412 MHz), width: 20 MHz (no HT), center1: 2412 MHz
        txpower 20.00 dBm
# iw dev mon0 del
```


Placing an interface into Monitor Mode - Linux
We can place an interface into monitor mode from the terminal in Linux. Let's take a quick look at the series of commands that we can use to do this.

iw dev wlan0 interface add mon0 type monitor

This command will take the existing wireless interface (wlan0) as a cue for the physical wireless device, then add a new interface called mon0. This interface is a monitor mode interface as represented by the 'type monitor'.

ip link set mon0 up

This command will take the new mon0 interface and put it into the UP state.

iw dev mon0 set channel 1

This command will take the mon0 interface and set it to channel 1 (2.4GHz Wi-Fi). We can repeat this command again to change channels. It is important to note that right now, mon0 is mapped to the same physical interface as wlan0, and settings channels on both could cause conflicts.

iw dev mon0 info

At any time, the iw info command can be run to pull information about the characteristics of the interface. We can see here that the interface is set to monitor mode, and channel 1 as we requested.

iw dev mon0 del

At any point we can remove the monitor mode interface.

While this combination of commands works to place the interface into monitor mode, it involves a lot of different command options. While we can look them up or try to commit them to memory, wouldn't it be easier to script this, as an activity we will do fairly often?

Placing an Interface into Monitor Mode – The Easy Way

Using the airmon-ng script included with the aircrack-ng suite, we can put an interface into monitor mode in a much simpler fashion (and change channels too!).

```
# airmon-ng
Interface       Chipset         Driver

wlan0           Ralink RT2870/3070     rt2800usb - [phy2]

# airmon-ng start wlan0
Interface       Chipset         Driver

wlan0           Ralink RT2870/3070     rt2800usb - [phy2]
                               (monitor mode enabled on mon0)

# iw dev wlan0 del
# iw dev mon0 set channel 1
```


Placing an Interface into Monitor Mode – The Easy Way

The aircrack-ng suite of tools has a number of uses for wireless analysis and attack. One utility included in the aircrack-ng suite is a shell script called airmon-ng. When airmong-ng is run without any arguments it will enumerate the wireless interfaces on the system. We can take this information (or, if we already know the interface name, we can skip this step) and issue a **start <interface name>**. This will create a monitor mode interface for us. Additionally, airmon-ng will check for other issues that may impact wireless stability, such as other processes that might interact with the wireless adapter and offer assistance in terminating those processes.

We can still use the iw command to set channels for our new interface, as noted above.

Aircrack-ng was written by Thomas d'Otreppe and a community of contributors and is available at https://www.aircrack-ng.org/.

Kismet

Kismet simplifies some of these activities for us:

- Puts interface into monitor mode
- Captures traffic of surrounding networks, and clients
- Aggregates that data into a simple, easy to view interface (and saves it all in a sqlite3 db for easy parsing!)

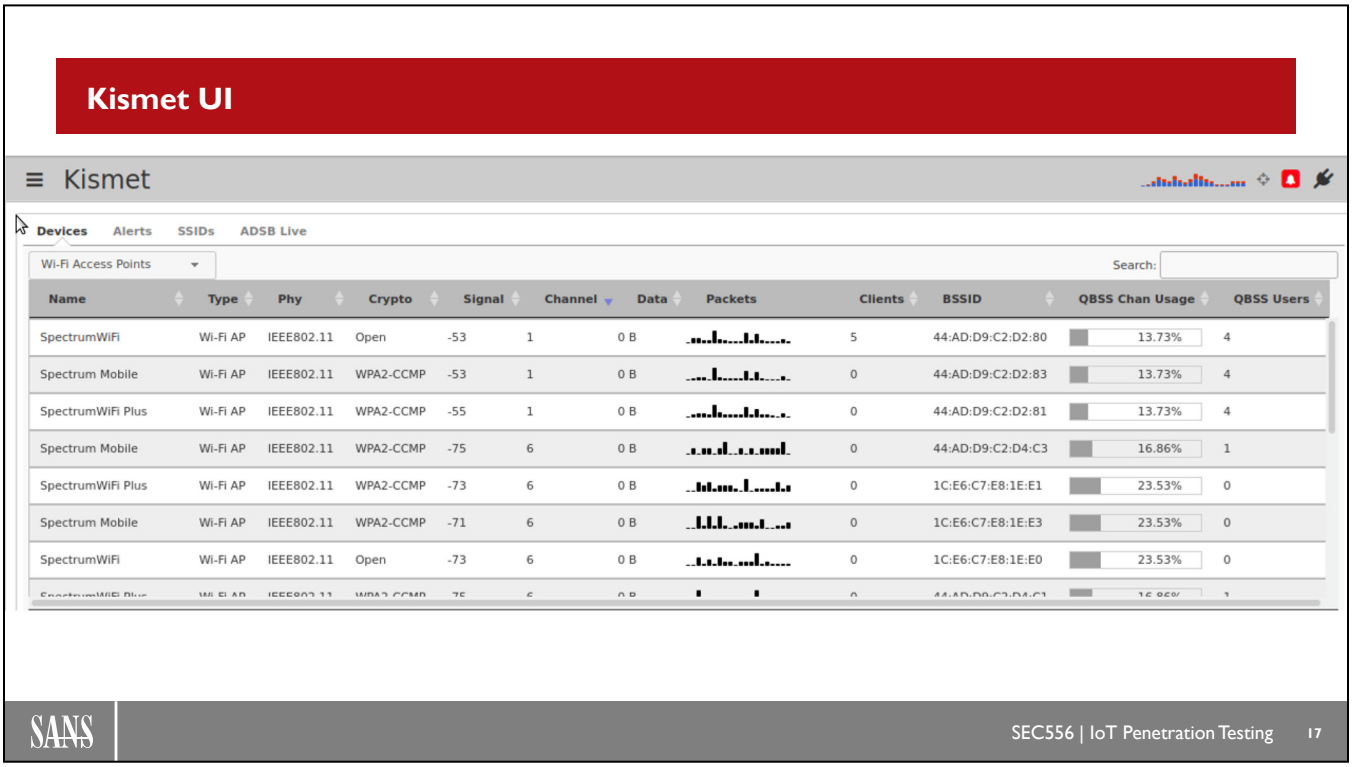
Kismet

Kismet rolls together a number of these features together, by taking wireless interfaces, placing them into monitor mode, channel hopping for optimal coverage, and capturing information about surrounding networks and clients.

Kismet also supports other technologies outside of Wi-Fi, allowing for discovery of Bluetooth Classic and BLE devices, with support for other devices being added as development continues.

Kismet outputs all the collected data into a unified sqlite3 database, however, data can also be exported into pcap, csv, and other formats.

Kismet is under heavy development and changes constantly. Kismet is developed by Mike Kershaw and a community of contributors and can be found at kismetwireless.net.



Kismet UI

The kismet UI is illustrated here. When launching Kismet, Kismet will create a web server to interact with the wireless interfaces. This local web server defaults to listening on port 2501.

Once you have used Kismet to configure an interface for capture, Kismet will start channel hopping and looking for Wi-Fi in your vicinity. At this point we can observe details of our surroundings like channel use, connected clients, supported encryption methods, and much more.

You will take a deeper dive into Kismet in Lab 3.1.

The MAC Layer

The MAC layer in 802.11 frames contains a plethora of information:

- Fragmentation support
- Separation of networks in the same frequency ranges
- 'Privacy' aka encryption
- Supported data rates

The MAC Layer

The Media Access Control (MAC) layer of Wi-Fi contains a wealth of information on the type of wireless frame we're dealing with, what capabilities are in use, and what type of wireless network we are interacting with.

Basic MAC Terms

IBSS (Ad-hoc) vs. Infrastructure networks
STA – station (client)
SSID – "name" of a Wi-Fi network
BSSID – MAC address of AP
ESSID – collection of APs with the same name
EAP – protocol for authenticating clients via 802.1X

Basic MAC Terms
Certain terms when dealing with the MAC layer may be unfamiliar to you. Let's cover some of the more common ones.

IBSS (Ad-hoc) vs. Infrastructure networks – These are two different types of Wi-Fi networks that achieve the same objectives via different means. We'll discuss these in more detail later in this section.

STA – Many references in these slides as well as the 802.11 specification will refer to the station or 'STA' for short. The station is, essentially, the client that is connected to the wireless network (usually via an access point).

SSID – Short for 'Service Set Identifier', this is what is colloquially known as the 'name' of a network. For example, the beacon frames from SANS-ROGUE01 will note its SSID as 'SANS-ROGUE01'.

BSSID – Short for 'Basic Service Set Identification', the BSSID is a 48-bit MAC address. In infrastructure networks, the BSSID is the MAC address of the wireless interface on the access point.

ESSID – Short for 'Extended Service Set Identifier', the ESSID is a aggregation of access points under the same SSID or friendly name. In this scenario, multiple access points will be serving traffic for a specific SSID. However, the BSSID for a given communication will match the MAC address of the access point you are interacting with.

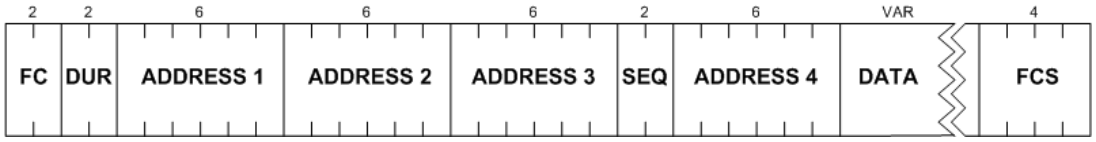
EAP – Short for 'Extensible Authentication Protocol', EAP allows for the authentication of a STA via 802.1X. This is the method used to authenticate individuals on a WPA or WPA2 Enterprise mode wireless network. However, every EAP type is different, and not all are created equally. We will discuss some examples of these in other modules.

802.11 Frame Header

The 802.11 frame header was defined (and modified) in 802.11 standards from 1999 forward.

Three types of frames are specified.

- Management
- Data
- Control

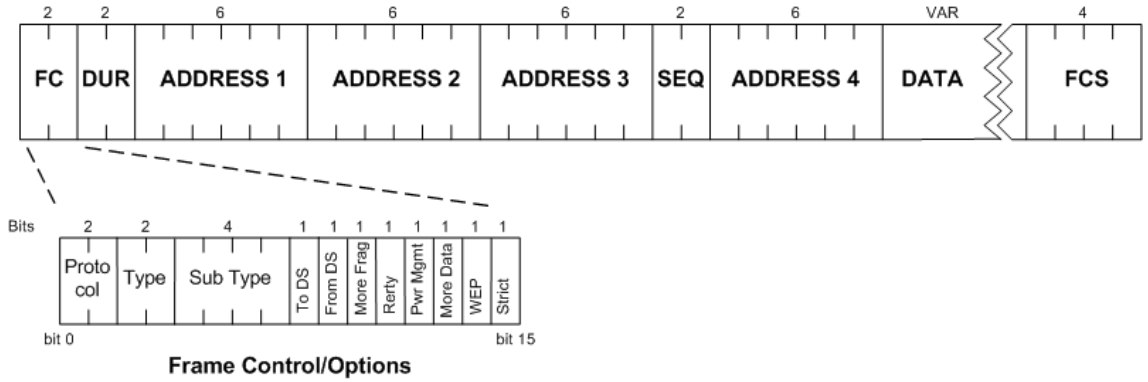


802.11 Frame Header

The 802.11 frame header is defined in the IEEE 802.11 specification. Three different types of 802.11 frame headers are specified in the 2 bit 'Type' area of the Frame Control Field:

- Management Frames – These frames assist with the overall functioning of the WLAN and include functions for requesting association/authentication to networks, probing for networks, and advertisements (beacons) from existing WLANs.
- Data Frames – These frames are used for the transmission of traffic (data) across the WLAN.
- Control Frames – These frames help with delivery of the other two frame types and include functions for medium management (RTS/CTS), ACK, and so forth.

802.11 Frame Control (FC) Field



Type and Subtype define the use/purpose of the frame
(type = 0 (management), type = 1 (control), type = 2 (data))

802.11 Frame Control (FC) Field
This is the typical layout of the Frame Control Field (2 bytes in size). There is one exception to this layout for Control frames with a subtype of 6, however, this is the layout you will almost always see.

According to the 802.11 specification, the protocol field should always be 0, and all other values are reserved.

The type/subtype values in the frame control specify the use and purpose of the frame – management, data, or control.

The To DS and From DS bits indicate if traffic is destined *for* the access point and systems beyond or is coming *from* the access point and systems beyond. These will be discussed in more detail in the next slide.

The more fragments flag indicate that the packet is fragmented, and more fragments are to come.

The retry flag indicates this frame is a retransmission of a previous one.

The power management field indicates the current status of power management on the STA.

The more data bit is used to help a STA from being put into power management when more data is en route.

The WEP bit is referred to in current versions of the 802.11 specification as the 'Protected Frame Subfield' and means the payload is obfuscated with some sort of cryptography – while this generally means WPA or WPA2 encryption at this time, this bit will also be set on WEP networks.

The strict bit is referred to in the 802.11 specification as the +HTC/Order subfield. This bit shows that this frame should (or should not) be strictly ordered, generally based on if Quality of Service (QoS) is enabled.

To DS and From DS Flags

To DS bit set = traffic going to wired network and AP
From DS bit set = traffic coming from wired network and AP
Both bits cleared = Ad-hoc network
Both bits set = WDS network

These affect use of the address fields in the header.

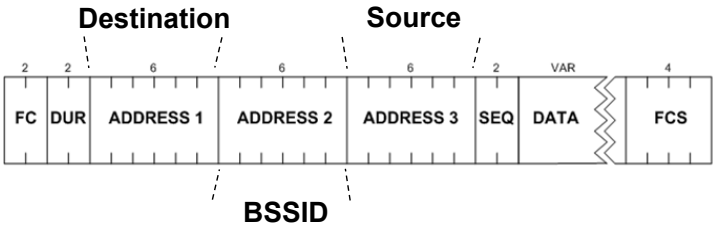
To DS and From DS Flags

The To DS and From DS flags have some important significance in the 802.11 header. While they show where data came from, and where it is going, they also determine the order and use of the address fields elsewhere in the header.

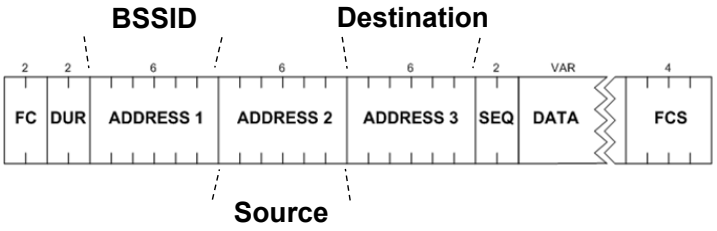
In most deployments, ad-hoc and Wireless Distribution Systems (WDS) are not typical. Therefore, the most common implementations will involve frames sent to the AP, with the To DS bit set, and frames coming from the AP, with the From DS bit set.

Address Field Usage

From DS bit set = traffic coming from wired network and AP



To DS bit set = traffic going to wired network and AP



Address Field Usage

When the From DS bit is set, the frame is going to the STA from the AP, and the destination, BSSID, and Source addresses are in address fields 1, 2, and 3, respectively.

When the To DS bit is set, the frame is going to the AP from the STA, and the BSSID, Source, and Destination are in address fields 1, 2, and 3, respectively.

The fourth address field is only used in WDS networks.

Common Management Frame Subtypes

Authentication request – authenticate to a WLAN
Association request – join a WLAN
Beacon frame – advertisements of AP
Probe request – STA looking for known WLANs
Deauthentication request – Disconnection request

Common Management Frame Subtypes
Several common management frame subtypes will be very prevalent in traffic of 802.11 networks. Some of these include:

- Association request – This is a request from a STA to join an existing WLAN (subtype 0).
- Authentication request – For networks that support authentication, this is a request to authenticate to a WLAN (subtype 11).
- Beacon frame – This is a frame used by an access point to advertise its presence and capabilities (subtype 8). This is probably the most prevalent management frame you will observe in most packet captures of wireless traffic.
- Probe request – When a STA joins a WLAN, the information about that WLAN is often saved by the STA Operating System. Periodically, the STA will look for WLANs that it has connected to in the past. How frequently this happens varies dependent on OS and revision (subtype 4).
- Deauthentication request – Access Points will sometimes request a STA disconnect temporarily for the purposes of load management. This request is a Deauthentication (often referred to as deauth) request and is subtype 12.

Management Frame Information Elements

These can specify various capabilities of the AP.

IEs in management frames are variable length.

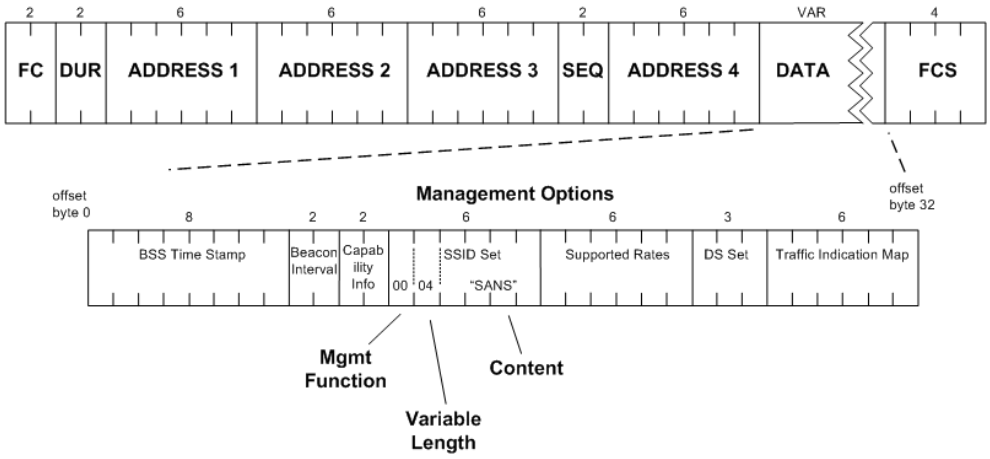
IEs are specified using well-known element IDs, and information fields.

A Management frame's purpose is defined in the data segment of the frame.

Management Frame Information Elements

In the frame body (data portion) a management frame can contain multiple fixed length pieces of information, as well as some variable length pieces known as Information Elements, or IEs. These IEs are well documented, and the capabilities described in the management frame can be readily decoded by tools like Wireshark.

Management Frame Information Elements Cont'd.



Management Frame Information Elements Cont'd.

Here is an example of a few information elements at work. In this case, the Information Element with an ID of '00' references the SSID of the network. Note here the tag of '00' and the length of the SSID field, so it can be decoded properly.

Let's Look at Encryption

Now we know how data gets from point A to point B, but how to protect it?

Encryption has been supported throughout the 802.11 lifecycle, but the options have evolved with the technology.

Solutions that protect data over wireless are not a substitute for technologies like VPNs, TLS, etc.

Let's Look at Encryption

While we have a basic understanding now of how a 802.11 frame is constructed, and what some of those elements mean, we do not know or completely understand yet how data can be protected from its source to its destination.

Over the years, the Wi-Fi Alliance has supported several standards in this space, and these standards have attempted to evolve with the industry and computing power. This endeavor has not always been successful, however.

In the upcoming slides, we will talk about the various encryption options to protect our wireless traffic.

NOTE: This discussion only covers the encryption of traffic as it transits the wireless network. Once it hits the access point, any of these encryption options are stripped away. As a result, it may make sense in many cases to rely on protocols for end-to-end encryption, such as VPNs or TLS.

Open Networks

Many hotspot networks as well as other providers may offer 'open' wireless networks.

- Emphasis on simplicity over security

These networks provide no inherent protection of data in transit over Wi-Fi.

Other protocols like TLS may protect your data.

If using an open wireless network, you must assume all traffic can be observed.

Open Networks

The first security option for encryption is…well, no security.

Many publicly offered wireless networks may not support encryption or other security options for ease of use. While this may be an effective solution for some, the problem here is that many people use these networks without completely understanding the risk.

Consider for example a mobile app on your phone. Under the covers of that mobile app, you may be utilizing many web services to send and receive data that is presented within the app. Not all of those web services may utilize TLS. This means that at least some of the data being sent to your mobile device may be unencrypted and sent over a wireless network where it is easily observed.

What is the risk here? It depends on the nature of the data being transferred over the wireless network.

Observation of the wireless traffic in an open network is trivial and can be accomplished with free software and most any computing system with a wireless card.

WEP

Originally released with 802.11, Wired Equivalent Privacy (WEP) was designed for a simpler time with less powerful hardware.

Uses RC4 (stream cipher) for encryption with CRC32 for integrity

Only encrypts the data payload

Unfortunately, the implementation of WEP had numerous challenges.

WEP

Wired Equivalent Privacy (WEP) dates back to the earliest days of Wi-Fi and was first detailed in the 802.11-1997 specification. WEP utilized standard ciphers for encryption (RC4) in conjunction with unique per packet values to generate unique key material. WEP shared secrets reside on all devices attached to a given network.

WEP only encrypted the data payload of the message and was only used on data frames (802.11 management and control frames were not impacted or changed by WEP).

Additionally, due to the lack of processing power on wireless equipment at the time, RC4 was supported in silicon for speed purposes, which became relevant once a replacement was sought for WEP.

Unfortunately, WEP had several design implementation flaws, which resulted in an overall lack of ability to use the protocol for its intended purpose of protecting data in transit over wireless.

WEP Issues

WEP relies on a per packet key – a combination of a shared secret and a unique per packet IV.

- However, the IV selection is cryptographically weak, and IVs are reused.
- This allows for the recovery of the shared secret with observation of less than 100k packets in most cases.

WEP allows for arbitrary 'blind' replay of an encrypted packet.

- This can be used to accelerate packet collection in quiet networks.

WEPs CRC32 checksum allows for byte at a time attacks to recover plaintext.

WEP is bad for multiple reasons; however, the capability to recover the shared secret via observation is probably the most dangerous issue, and the most readily used.

WEP Issues

WEP leverages a per packet key that is a concatenation of the shared secret and the IV. The problem here is that the selection of the IV is a cryptographically weak process, and over time certain 'weak IVs' emerge that can allow for cryptanalytic attacks against the shared secret itself.

This type of attack is very successful and only requires a certain number of observed packets to be effective. In almost all cases, less than 100k packets from a WEP network will allow for retrieval of the WEP shared secret.

WEP also does not enforce sequencing of packets, meaning packets can be captured and replayed. This can allow for acceleration of packet collection for the above attack in networks with less communications.

Finally, the CRC32 checksum for integrity allows for single byte manipulation or 'chop-chop' attacks to retrieve the plaintext of a message.

While this is several attacks against WEP, the ability to recover the shared secret via passive observation is the most readily effective and fastest method of attack in most cases.

WEP Shared Secret Recovery in Action

```
$ aircrack-ng -n 64 aircrack-data.dump
Read 152633 packets.
Choosing first network as target.
Starting PTW attack with 75811 ivs.

                                Aircrack-ng 1.0 beta2 r1069

                [00:00:15] Tested 1665641 keys (got 8996 IVs)

   KB    depth   byte(vote)
    0   84/ 85   FE(9728) 15(9472) 18(9472) 1C(9472) 25(9472)  40)
    1   22/ 25   92(11264) 17(11008) 41(11008) 69(11008) 73(11008)
    2   19/  2   F3(11520) 2E(11264) 42(11264) 44(11264) 75(11264)
    3   22/  3   ED(11264) 67(11008) 69(11008) 97(11008) F6(11008)
    4   15/ 16   02(11520) 36(11264) 8A(11264) DB(11264) 07(11008)

                        KEY FOUND! [ 88:85:74:27:35 ]
```


WEP Shared Secret Recovery in Action

In this example, we are performing an attack to recover the shared secret of a WEP network using the aircrack-ng utility. Because of the relatively low computational overhead of the attacks against WEP, this is a brute-force attack against the shared secret, so no wordlist is required.

Note, this is a slightly older screenshot of the PTW attack, but it was successful against a WEP network, yielding the shared secret, in 15 seconds.

WEP Attack Impacts

Once the shared secret is obtained, we can connect to the network, or decrypt traffic with tools like airdecap.

```
$ airdecap-ng -w 88:85:74:27:35 aircrack-data.dump
Total number of packets read        152633
Total number of WEP data packets    101145
Total number of WPA data packets         0
Number of plaintext data packets         4
Number of decrypted WEP  packets    101082
Number of corrupted WEP  packets         0
Number of decrypted WPA  packets         0
```


WEP Attack Impacts

Once we've been able to obtain the shared secret for a WEP network, we have unfettered access to the network. We can join that network and act as a node on the network, observing all data as decrypted content. Or we can decrypt previously captured traffic using the shared secret as well.

Final Thoughts on WEP

WEP has plenty of issues and is the oldest technology for encryption of 802.11 network traffic.

However, we still see it in the wild periodically.

When you do see it, confirm your scope because the WEP network often has a high value target, that may be brittle.

- Proceed with caution, but potentially high rewards ahead!

Fixing WEPs issues actually became the design goals for its successor.

Final Thoughts on WEP

While we've covered it at a fairly high level here, WEP has a number of issues that mean we should never see WEP in production use. That being said, this author will encounter WEP several times a year in his engagements.

There can be any number of reasons for this. One of the most common is that a business decision is made to keep a WEP network around to service a very old piece of equipment that has been deemed mission critical. In many cases, this equipment can't be readily replaced or uplifted to new technology and only supports WEP, leaving the company to build a network solution around their hardware problem.

If you encounter one of these WEP environments in your testing, you have a great opportunity to bring value to the business by showing the danger of this kind of legacy implementation being left out in production. However, when interacting with this network and any legacy devices, take care to not test them too aggressively, as older technology can be brittle, and become unstable from something as simple as a nmap scan. Finally, confirm such a device and network are in scope for your test.

WEPs implementation flaws were well known and drove the adoption of the next Wi-Fi encryption solution…WPA.

WPA

WPA had to fix WEPs IV reuse, CRC32 checksum, and replay vulnerabilities.

- And it had to continue to use RC4 for encryption due to the hardware of that time.

WPA was designed to be an interim solution to bridge the gap to WPA2 – however, we still see it in use at times.

- Even more common is to find WPA2 networks deployed in a 'compatibility mode', which makes that network vulnerable to most WPA attacks.

WPA

WPA had to address the shortcomings of WEP, while being able to run on the same hardware. Most Wi-Fi hardware at the time (early 2000s) had relatively meager processing power and was quite expensive. The decision was made to develop a 'drop-in' solution for WEP on existing hardware and develop a more fully vetted solution with the time bought with the interim solution.

The drop in solution was WPA. The long-term solution was WPA2. WPA was intended to be phased out after WPA2 was introduced; however, WPA implementations still pop up here and there. More importantly, WPA2 was developed with a 'backwards compatibility' mode that allows WPA devices to interact with a WPA2 network. Unfortunately, this opens the WPA2 network up to additional WPA-only attacks.

More WPA

WPA can be deployed in pre-shared key (PSK or Personal) mode or Enterprise mode.

- These specify the authentication process; encryption of traffic remains the same.
- PSK authenticates via a shared secret on every device.
- Enterprise mode authenticates via the Extensible Authentication Protocol (EAP), and authentication methods can vary.
- Most IoT devices will use PSK mode networks.

More WPA

WPA has two modes of deployment:

Personal or PSK mode is the most common mode of deployment in consumer and small/medium business environments. Enterprise mode is more complex to deploy and maintain, and as such, is seen less frequently outside of enterprises.

Personal vs Enterprise mode are largely differentiated by the process used to authenticate a client. In Personal mode, the client is authenticated by a Pre-Shared Key (PSK), a shared secret present on every device in the network. In Enterprise mode, authentication occurs via an implementation of the Extensible Authentication Protocol (EAP). Every EAP type can vary, but common authentication methods include userid/password, MFA, Digital Certificate, and others.

Due to the relative ease of deployment and lower overhead on the endpoint device, IoT devices almost always connect to Personal (PSK) networks, and they are what we will focus on here.

WPA – Fixing WEP

WPA corrected the following WEP vulnerabilities:

- Replay attacks – added a per packet sequence counter
- CRC32 – replaced with a bespoke message integrity check algorithm called Michael
- IVs for entropy– mitigated with key rotation via a mutual 4-way handshake

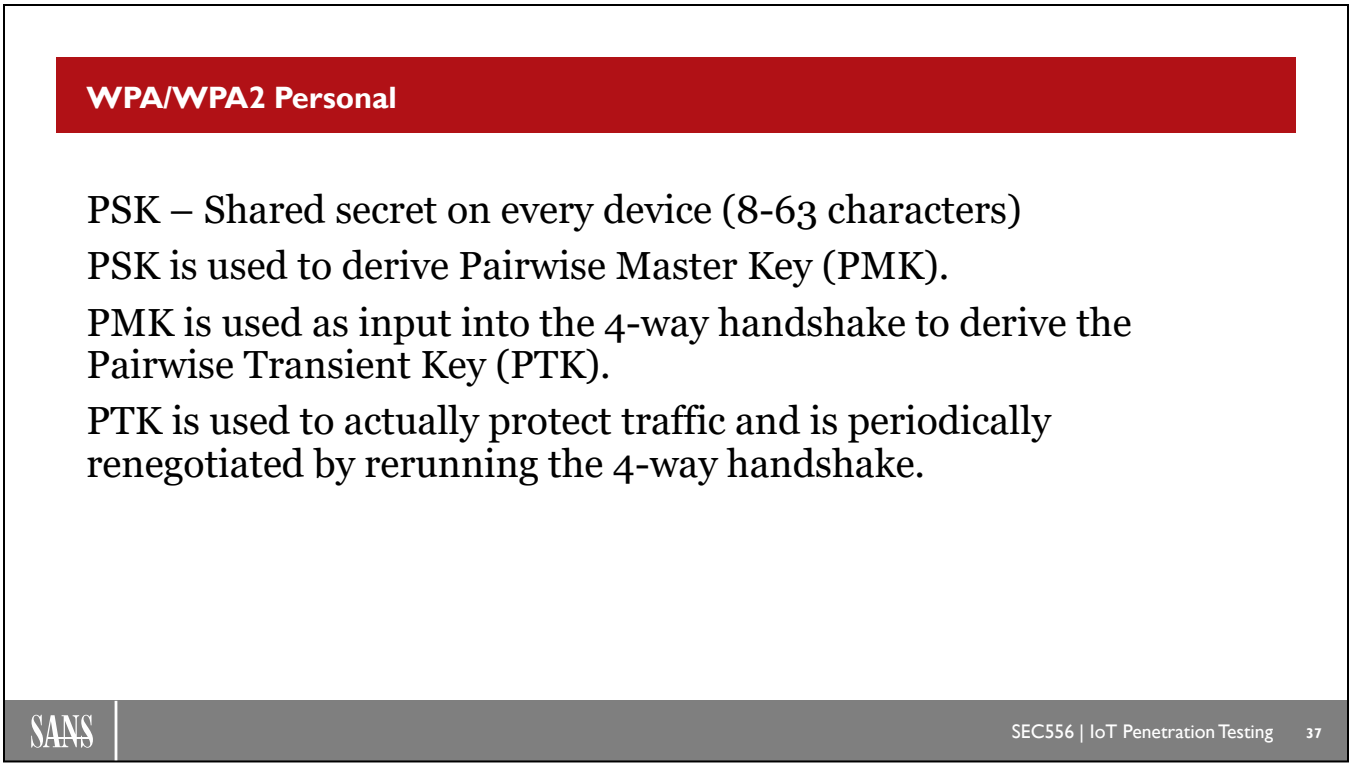
WPA – Fixing WEP

WEP had 3 significant vulnerabilities addressed by WPA:

Replay attacks – you will recall our packet capture acceleration attacks by retransmitting packets back to the network to generate new packets, and new IVs. This was mitigated in WPA by adding a sequence counter that incremented per packet and is checked at the destination to ensure the counter has incremented from the last packet received from that device.

Weak CRC32 checksum – this was mitigated by a new Message Integrity Check algorithm. This algorithm actually enforced strict penalties for packets that were modified in transit and failed the integrity check.

IV reuse – This was mitigated by frequent key derivation and rotation, via a new key derivation process called the 4-way handshake.



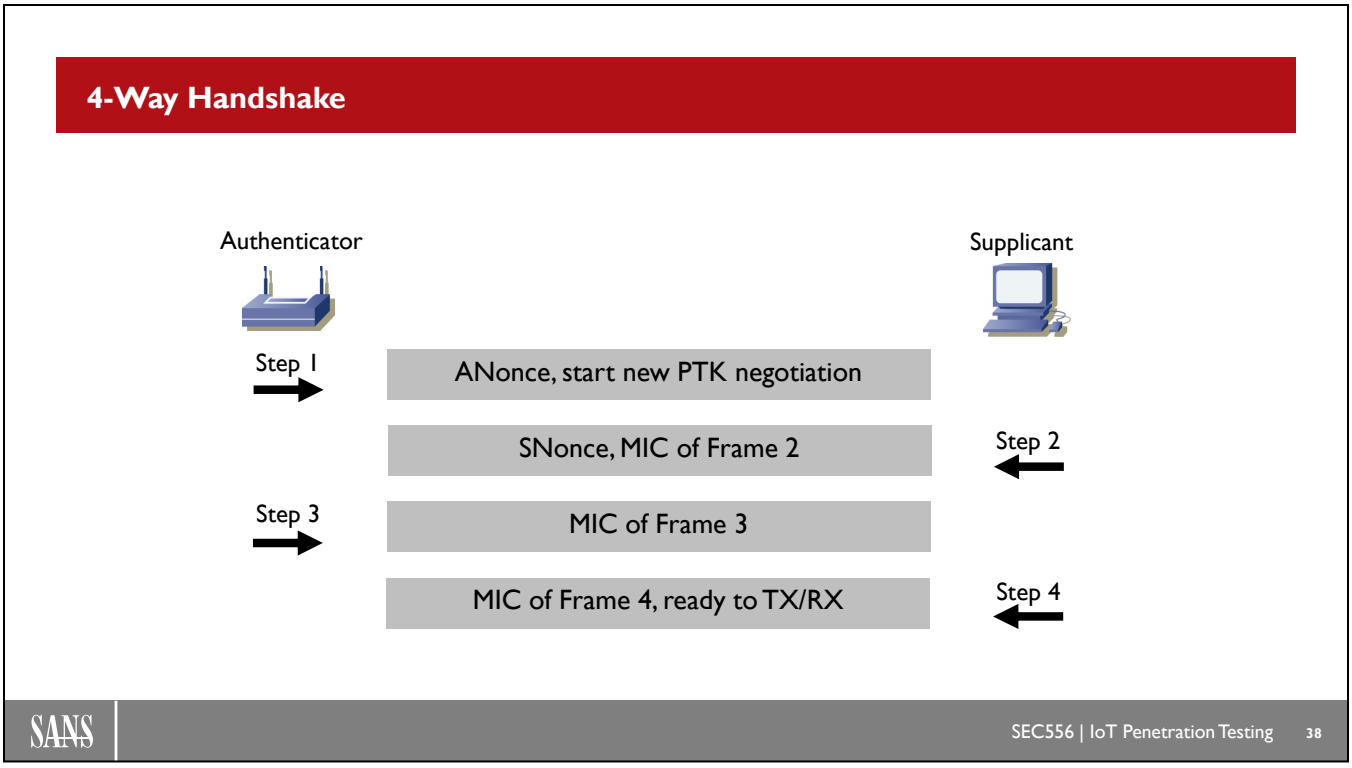
WPA/WPA2 Personal

Personal mode (for both WPA and WPA2) has LOTS of keys used in various parts of the process.

The first is the Pre-shared key, or PSK. The PSK resides on every device that attaches to the network and is used to authenticate that device to the network. This is an 8 to 53-character passphrase.

The PSK is used, along with the SSID (friendly name) of the network, and the length of that SSID (len(SSID)) are used to derive the Pairwise Master Key (PMK). The PMK is meant to be static and will only change on a device if the PSK for the network is rotated. The generation of the PMK is designed to be computationally expensive, to resist the potential for brute force attacks.

The PMK is used to derive the Pairwise Transient Key (PTK) via a process called the 4-way handshake. The 4-way handshake was designed to ensure that both client and access point contribute to the key derivation process, and validate the other device has the same key material before beginning communications.



4-Way Handshake
The WPA2 4-way handshake is the method where the supplicant (client) and the authenticator (generally the access point) exchange the necessary information to derive the Pairwise Transient Key (PTK) information.

Before the 4-way handshake, it is assumed that both stations have knowledge of the Pairwise Master Key, either derived from the pre-shared key (PSK) in WPA2-PSK networks or as the result of an 802.1X protocol exchange. Both clients are assumed to have knowledge of their own MAC address and the MAC address of the other side of the connection.

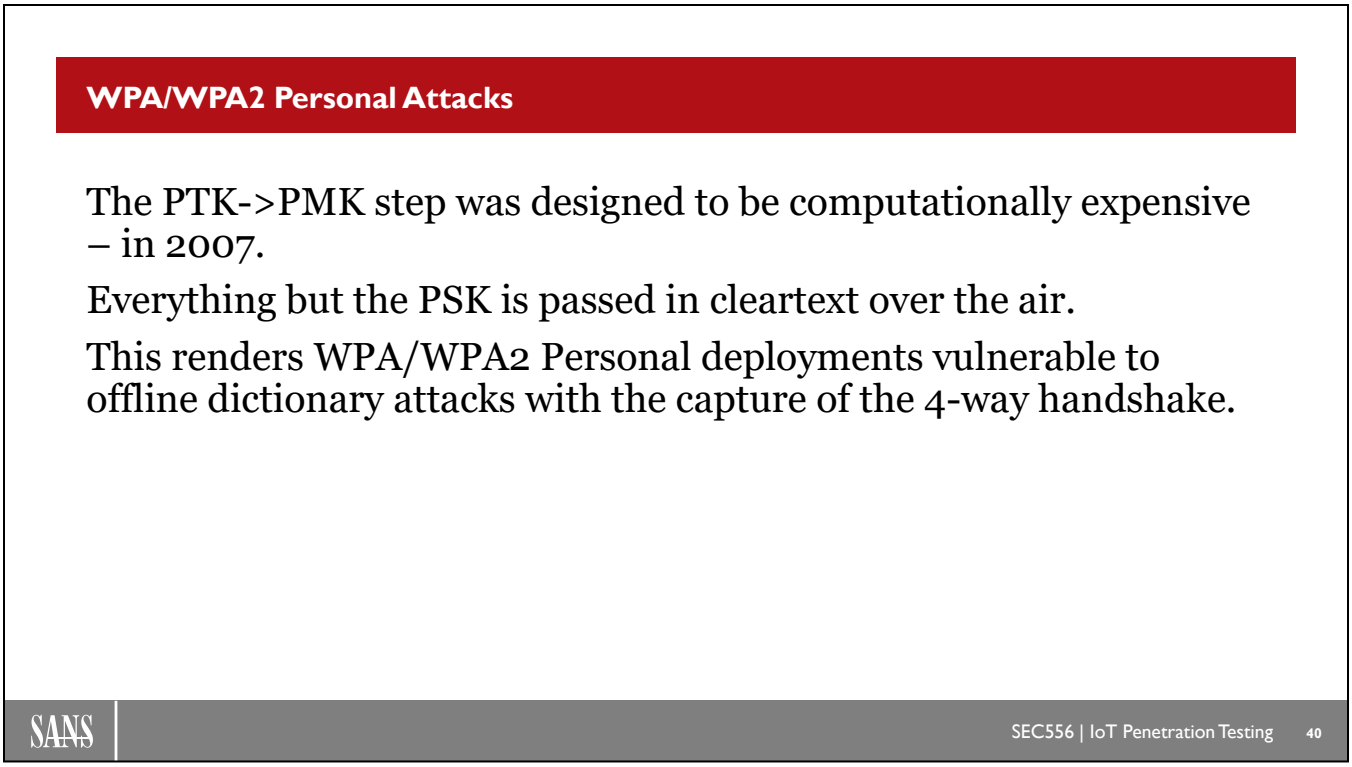
Message 1 in the 4-way handshake delivers a frame from the authenticator to the supplicant that begins the PTK negotiation. This frame contains the first of two nonces—the authenticator nonce (ANonce), used in the PTK calculation. The first frame does not carry a message integrity check (MIC) since the authenticator does not have enough information to generate the PTK (only one nonce is known) and therefore does not have a MIC key value needed for the calculation.

Message 2 originates from the supplicant and contains the supplicant nonce and the MIC of the frame. Before transmitting this frame, the supplicant has knowledge of the supplicant nonce and authenticator nonce information and can calculate the PTK. Once the PTK is calculated, the HMAC MIC key is known, and it is used to seed the HMAC-SHA1 or HMAC-MD5 algorithm to generate a 128-bit MIC value. When the authenticator receives Frame 2, it extracts the supplicant nonce and is able to calculate the PTK using the calculated HMAC MIC key to validate the MIC of the packet.

Message 3 originates from the authenticator and includes only the MIC calculation of the packet. This frame validates to the supplicant that there is no machine-in-the-middle attack underway. The verification of the MIC by the supplicant verifies that the frame has not been modified and that the authenticator has successfully calculated the PTK (since using an incorrect PTK would lead to an incorrect HMAC MIC key, which would generate an invalid MIC calculation).

Message 4 originates from the supplicant to acknowledge the receipt of Frame 3 and to indicate that the supplicant is ready to transmit and receive data.

Note that the nonce values are critical in this exchange since they protect the protocol from replay protection. While traditional replay mitigation would make use of an explicit sequence number that is included in a frame and acknowledged by the receiving party, the WPA2 4-way handshake uses an implicit check to ensure the nonce values are unique and have not been modified in transit by mixing them into the PTK calculation. Both nonce values are verified, due to the explicit MIC calculation on Frames 2–4. If either nonce was incorrect, the MIC calculation would fail since the PTK would be different.



WPA/WPA2 Personal Attacks

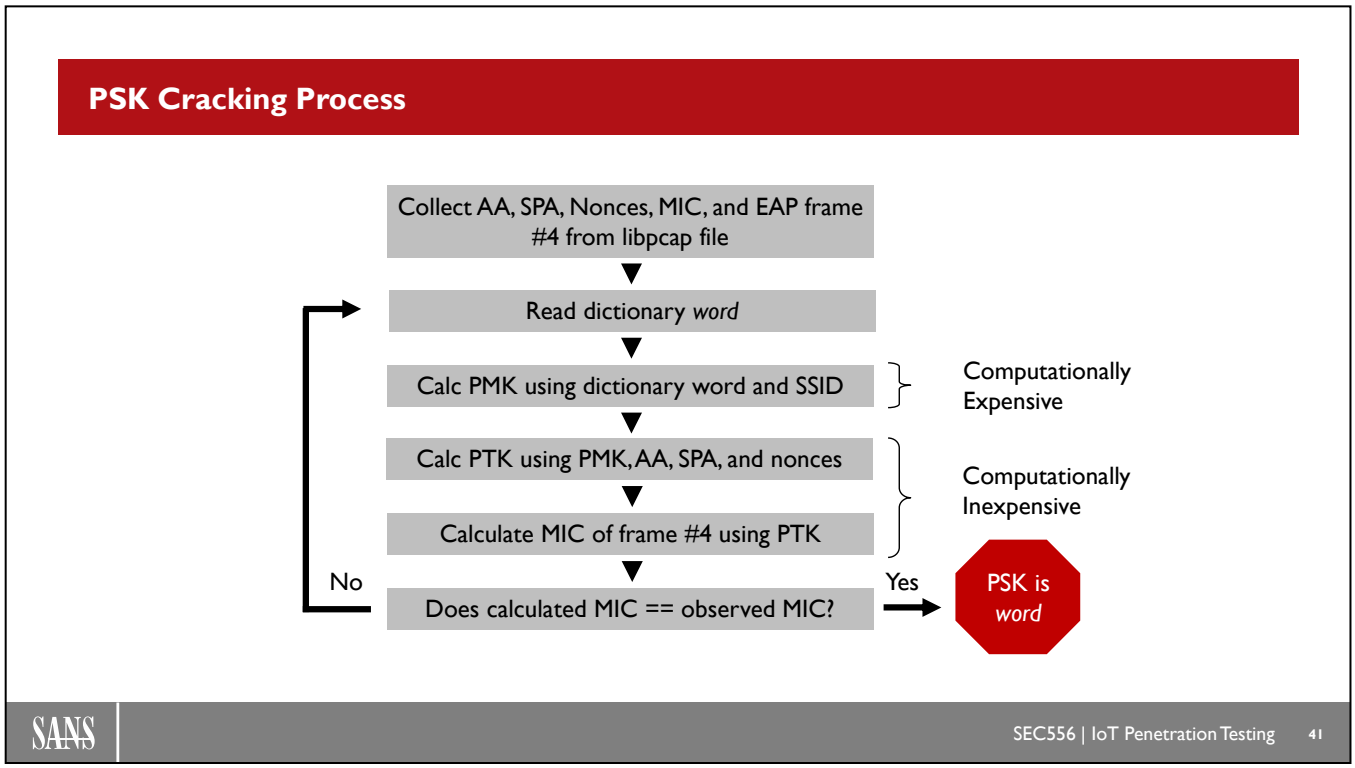
You will recall that the PSK -> PMK step is designed to be computationally expensive, to prevent brute forcing. If you have joined a WPA/WPA2 Personal network from a mobile device or a game console, you have likely noticed a 5-10 second delay between submitting the PSK and getting a notification that things were complete. This is that computationally expensive step happening.

However, as great and as fast as our mobile devices and gaming consoles are, they do not compare to the raw computing power of modern GPU's or cloud infrastructure.

Simply put, the computational expense of the PSK->PMK step was very effective against brute force attacks in the mid-2000s when it was introduced, but the continued improvement of computing power has definitely lowered the bar for this kind of attack.

The collection of the 4-way handshake process allows for offline dictionary attacks to determine the PSK. All elements of the PSK->PMK->PTK process are transmitted over the air, unencrypted, with the exception of the PSK itself. So, we can populate a wordlist and attempt them through the process to see if we have the right PSK in our wordlist.

We do have one advantage that mitigates this risk somewhat – the PSK->PMK process uses the SSID, which means an adversary cannot pre-compute the wordlist to PMK candidates for faster cracking – if they do not know the SSID in advance!



PSK Cracking Process

This slide documents the steps used in PSK cracking. The first step is to collect all the information needed from the libpcap file to calculate the PTK, including the authenticator MAC address, the supplicant MAC address, the supplicant nonce, and the authenticator nonce. These fields are needed as input criteria to calculate the PTK using the PBKDF2 algorithm.

Additionally, we need to verify if the PTK calculated from the dictionary word is the correct PTK. The fourth frame of the 4-way handshake is needed for this check. We use the contents of the fourth frame to calculate the MIC from the current dictionary word and PTK. If the calculated MIC matches the actual MIC, we know the current dictionary word is the correct passphrase.

Once all the processing data has been collected, the second step is to read a dictionary word. Next, the PMK is calculated using the dictionary word and the SSID for the network supplied on the command line. This step is the most computationally expensive step, requiring many iterations of the SHA1 hashing protocol in the PBKDF2 algorithm.

Once the PMK has been calculated, the PTK is calculated using the PMK, AA, SPA, and nonces as input data. Next, the PTK is used to identify the HMAC MIC key, and an HMAC-MD5 operation is completed on the payload of the fourth frame in the 4-way handshake. If the calculated MIC matches the MIC seen in the capture file, we know the PTK is correct and therefore the dictionary word is the correct passphrase. If the calculated MIC does not match the MIC stored in the libpcap file, we read the next dictionary word and continue the auditing process until we get a matching MIC value, or we run out of dictionary words.

Dictionary Attacks

Many tools exist for dictionary-based attacks of WPA/WPA2 personal shared secrets.

- Hashcat
- Aircrack-ng
- John

Cracking is also possible via cloud-based infrastructure.

- We still depend on wordlists though!

Hashcat also supports mask attacks; a brute force attack where part of the shared secret is known, or the construction (character set) is known.

Dictionary Attacks

Dictionary based attacks against the 4-way handshake were first implemented in tooling by Joshua Wright via a tool known as coWPAtty. The capabilities of this tool were folded into aircrack-ng where multithreading was added.

Later, the same capabilities were added to john and hashcat. This has greatly accelerated the speed with which wordlist testing can be performed, both via computational improvements and the leveraging of GPUs when possible. However, in all these cases, we are still relying on the word to be present in the wordlist somewhere.

One exception to this is hashcat, which supports a mask attack mode. Mask attacks leverage partial knowledge of the PSK, whether it be construction style, length, or anything else that reduces they keyspace such that brute force attempts become feasible. As one example, consider a cellular portable hotspot that broadcasts a WPA2-PSK network for clients to connect to. This network uses a randomly generated password of 8 characters using the alpha and base10 number keyspace (a-z, A-Z, 0-9). This is a significant reduction of the keyspace, and brute forcing becomes possible. How could we come to this determination? We could observe a few hardware samples and look for common construction between them.

We can also leverage the power of the cloud for high powered machines at lower cost, or for distributed cracking. Coalfire has created a tool called NPK that implements this concept at https://github.com/Coalfire-Research/npk.

John the Ripper (aka john) is created by Solar Designer and a community of contributors, and is available at https://github.com/openwall/john.

Hashcat is created by Jens Steube, Gabriele Gristina, and a community of contributors, and is available at https://github.com/hashcat/hashcat.

WPA/WPA2 Personal Attacks Cont'd.

Sometimes all steps of the 4-way handshake can be problematic to capture.

- Signal issues, heavy traffic, etc.

Another method to perform PSK cracking was identified in 2018 that does not require capture of the full 4-way handshake – only frame 1.

- PMKID, created to support roaming implementations
- Not all implementations support this but can be identified via Kismet when a client joins the network.
- Dictionary attacks to recover the PSK are still required.

WPA/WPA2 Personal Attacks Cont'd.
In many situations, it can be difficult to capture all 4 steps of the 4-way handshake, usually due to signal interference, but also range, etc...

In 2018, the Hashcat team identified and published a new attack against the 4-way handshake. This relies on a value called the PMKID, which is published when certain APs have fast roaming enabled. The PMKID is transmitted with the first frame of the 4-way handshake.

The PMKID is derived as follows:

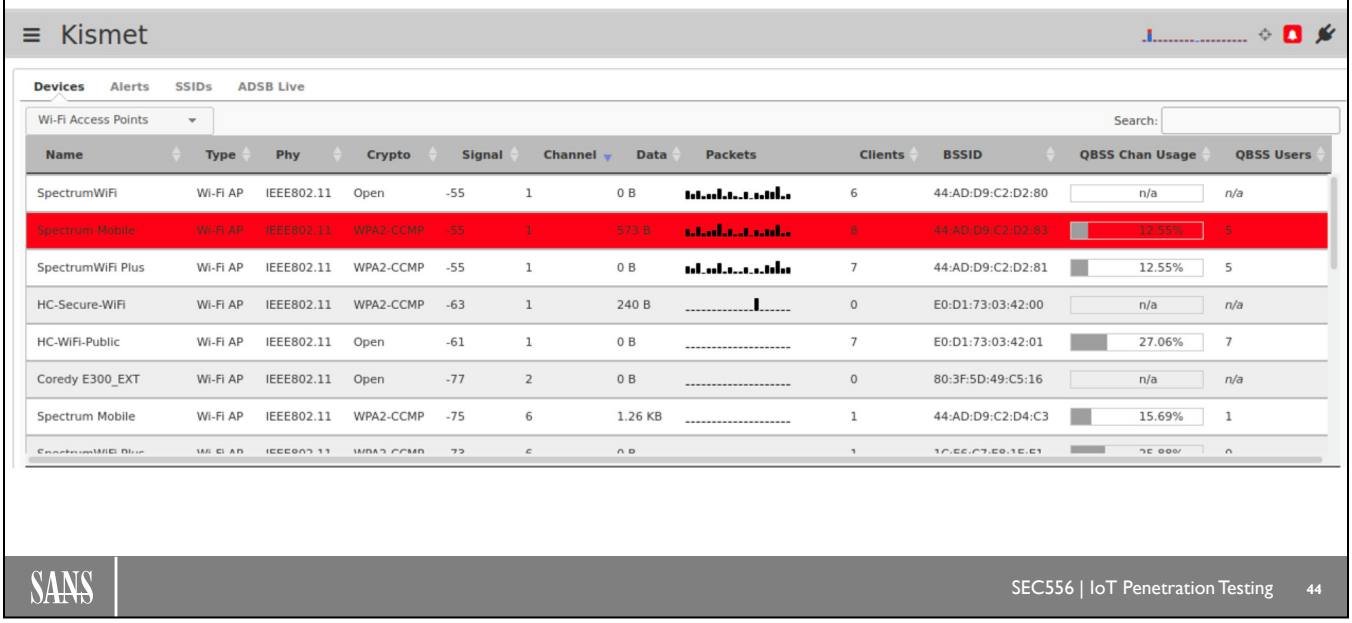
PMKID = HMAC-SHA1-128(PMK, "PMK Name" | MAC_AP | MAC_STA)

(In this case "PMK Name" is a static phrase).

The presence of the PMKID means we can perform wordlist-based cracking against this network without having the full 4-way handshake.

Kismet can identify the 4-way handshake AND the PMKID if used and write them out to pcap files for use in cracking.

Kismet for PMKID/4-Way Identification



Kismet for PMKID/4-Way identification

If Kismet observes a PMKID in use, or a normal 4-way handshake, it will highlight that network, and give you an option to download pcap files with only the PMKID data/4-way handshake data. These can be passed to hashcat for cracking after some conversion with the *hcxpcapngtool* utility.

WPS for PSK Distribution

The Wi-Fi Alliance determined that entering an 8-character password on some devices was too hard.

- Low function/non-existent HMI

Wi-Fi Protected Setup was created by the Wi-Fi Alliance as an alternative means to authenticate in WPA/WPA2 personal networks.

- Enter a PIN
- Push a button on AP
- USB/NFC (optional)

After successful enrollment with WPS, the PSK is provided to the client.

WPS for PSK Distribution

Sometimes, well meaning decisions can go awry. That happened with WPS.

Wi-Fi Protected Setup, or WPS, is a standard from the Wi-Fi Alliance that allows a client to shortcut the process of entering a PSK. Instead, the authenticate via a different means, then the PSK is provided to them.

The WPS authentication methods are:

- PIN Entry – enter an 8-character PIN.
- Push-button – similar to configuring a rolling code device, pushing a button on the AP will allow any device that attempts to connect to the AP for the next 60 seconds to connect successfully.
- USB/NFC based pairing for client – these methods are optional in the WPS specification.

WPS Cont'd.

This process allows us to bypass the security of the PSK.

- The PIN is only 8 characters.

That is treated as two parts and evaluated separately

- Remember LANMAN hashing??

This reduces the overall entropy of the solution allowing for brute force PIN guessing.

- Practical implementations of this exist in reaver (online attack) and pixiewps (offline attack).

WPS Cont'd.

If you are thinking that an 8-character PIN could subvert the security of a (potentially much longer) PSK, you would be correct. But it's actually worse than that.

The WPS 8-character PIN is actually split into 2 4-character parts and evaluated separately. This greatly reduces the entropy of this solution, which means that a WPS PIN can be guessed within 11000 or so guesses, assuming no other countermeasures are in place.

If this sounds familiar, you may be familiar with the LANMAN hashing method on Windows, which took a similar approach to secret storage (albeit with a different number of characters in the two parts).

A practical implementation of an online PIN guessing attack was implemented with reaver (https://github.com/t6x/reaver-wps-fork-t6x). Some AP manufacturers have developed countermeasures to try and thwart PIN guessing attacks.

An additional vulnerability is present in some APs that allow for an offline attack against WPS. This is known as the pixie-dust attack and is implemented in pixiewps (https://github.com/wiire-a/pixiewps).

WPA3

WPA3 is the newest iteration of wireless security.

Design goals include:

- Mitigation of current WPA2 4-way handshake attacks with a new handshake process (SAE, Simultaneous authentication of equals).
- Support for encryption over unencrypted networks when supported (OWE, Opportunistic Wireless Encryption).
- Mitigation of WPS attacks with a new Device Provisioning Protocol (DPP).
- Protected Management Frames *required* (mitigates deauth-style attacks).

WPA3

WPA3, while relatively new, has been long expected to address some of the common vulnerabilities in WPA/WPA2 implementations:

-A new handshake process called SAE (Simultaneous authentication of equals) which mitigates some of the current cracking attacks

-Support for encryption when both AP and client support it over an otherwise open network

-New version of WPS called the device provisioning protocol (DPP)

-Finally, WPA3 requires Protected Management frames, which limits the effectiveness of deauth-style attacks.

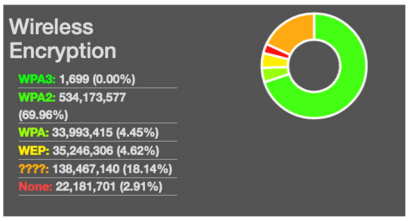
However, not all of these requirements *must* be supported for a product to be called WPA3 – just the new handshake.

WPA3 in IoT

Due to WPA3 implementations requiring new hardware, it is not typically seen in IoT deployments (or generally) at this time.*

It is possible to deploy WPA3 networks in a backwards compatibility mode to support WPA2 clients (WPA2 supports the same thing for WPA).

This subjects the WPA3 network to all the traditional risks and attacks associated with a WPA2 deployment.



*June 2021 WiGLE summary of observed networks by encryption

WPA3 in IoT

Unfortunately, WPA3 requires new hardware, and as such, uptake in IoT has been pretty much nonexistent to this point. In fact, WPA3 implementations overall have been fairly slow.

As of this writing, Wigle.net (wardrive aggregator) is only showing a few thousand WPA3 networks in the wild, as opposed to half a billion WPA2 networks.

While these numbers will be skewed due to how long WPA2 has been around, the WPA3 number is still very low when you consider WPA3 hardware has been available for about two years.

As such, most of your effort as testers should be spent on WPA2.

Practical Attack Flow – Wi-Fi – Recon

With this, we can start to envision a basic attack workflow against a Wi-Fi IoT device.

1. Perform monitor mode capture.
   a) Objectives: Identify type of encryption in use, targeted network
   b) Bonus objectives: gather additional candidates for wordlist, commonalities of PSK construction based on vendor/product, handshakes, PMKID
   c) Impact: Information disclosure for unencrypted networks

Practical Attack Flow – Wi-Fi – Recon

Given all this information, we are probably starting to come up with a plan of attack against a network with IoT devices of interest to us.

We should start with passive discovery of assets, via monitor mode capture. Identify the networks of interest, and verify they are in scope with your testing point of contact as needed.

Once you identify your target networks, lock channels and take an extended look at them. You may also start thinking about ways to enhance your attacks against the deployment if it's a PSK network – look for terms and features that may enhance your wordlists. Also, if you can identify the vendor, look for common or default credentials for these devices, or PSK construction schemes that may lend them to mask attacks. Grab 4-way handshakes if you can, or PMKIDs if these are available.

Don't forget, if you have unencrypted networks like BYOD in scope, evaluate their traffic for information disclosure issues. If you can see sensitive traffic, so can an adversary.

Practical Attack Flow – Wi-Fi (WEP) - Attack

Our next step depends on the network type.

2. WEP
   a) Objectives: Obtain WEP key (shared secret)
   b) Techniques:
      i. Passively sniff WEP traffic, brute force crack using aircrack-ng
      ii. Retrieve device attached to the network, extract the key (wirelesskeyview, netsh, keychain, wpa_supplicant.conf…)
   c) Impact: Join network and/or decrypt traffic

Practical Attack Flow – Wi-Fi (WEP) – Attack

If we are dealing with a WEP network, we should try and obtain the WEP key to join the network or capture and decrypt its traffic.

Passively capture WEP traffic for this network, and once you have enough, go ahead and brute force the WEP key using a tool like aircrack-ng. Patience pays off here, as long as there is traffic, you will get there – be patient.

While you can accelerate capture with packet capture and replay, this can also cause unexpected results at the AP especially if you do not rate limit your replays. This can cause business impact, which you should avoid when possible.

Additionally, if you can obtain a device that can communicate on the network, you can try to log into it with default credentials or other methods and extract the WEP key that way.

Practical Attack Flow – Wi-Fi (WPA/WPA2) - Attack

Our next step depends on the network type.

2. WPA/WPA2 Personal
   a) Objectives: Obtain WPA key (shared secret)
   b) Techniques:
      i. Passively sniff WPA traffic, capture 4-way handshake, wordlist-based cracking via hashcat
      ii. Retrieve device attached to the network, extract the key (wirelesskeyview, netsh, keychain, wpa_supplicant.conf…)
      iii. If enabled, use WPS to recover the PSK (bruteforce or pixiedust methods)
   c) Impact: Join network and/or decrypt traffic

Practical Attack Flow – Wi-Fi (WPA/WPA2) – Attack

If we are dealing with a WPA or WPA2 Personal network, then we have a slightly different approach. Brute forcing like we used for the WEP key won't work here. But we still want the PSK to access the network or decrypt its traffic.

We start the same way, passively capturing traffic. We're looking for 4-way handshakes or PMKIDs we can pass to wordlist-based cracking. If you have any way to limit the keyspace using vendor defaults, brute force may be an option.

Given some of the uncertainty of this approach, it may be significantly easier to try and access a device with access to the network and extract the PSK from the device. This approach will vary depending on the device and OS.

Finally, if WPS happens to be enabled on the network, it is definitely worth attempting to access the PSK using reaver or pixiewps style attacks.

Practical Attack Flow – Wi-Fi – Profit

3. Profit
   a) Objectives: Disrupt operations, other Red Team objectives as appropriate
   b) Techniques:
      i. Use network access to retrieve information/inject false information
      ii. Use decrypted information from target networks to further other objectives
   c) Impact: Unknown

Practical Attack Flow – Wi-Fi – Profit

What we do next with the information we have obtained in the previous steps depends on the engagement. If this is a more traditional penetration test, this may be sufficient information for your report, though it may be worth seeing if these wireless networks you have access to allow access to other network resources of interest. If this is more of a Red Team engagement, you may want to use your access to these networks to try and access other information, inject false information into networks, or use the (previously encrypted) wireless information to further your objectives on other fronts.

Summary

Wi-Fi remains a popular option for devices with mains power sources.

Unfortunately, most IoT devices lack support for more robust security options like WPA3 and Enterprise mode authentication.

This coupled with the nature of shared secrets make Wi-Fi for IoT difficult to protect.

Consider SANS SEC617 for more Wi-Fi goodness.

Summary

In this module, we started to scratch the surface of Wi-Fi. We discussed some of the essentials, as well as the more common encryption types and their strengths and weaknesses.

At this time, since most IoT devices do not support more robust security controls like WPA3 or Enterprise Mode WPA2, we are forced to consider 10-year-old (or older!) techniques to interact with these devices and networks.

One important consideration is that all of the methods we've discussed use a shared secret to authenticate the device. This makes protection of the network very difficult, and our job as penetration testers easier for initial exploitation, if not remediation.

If you found this module interesting, we spend over 2 days on Wi-Fi in SANS SEC617, as well as other types of wireless as well.

Exercise: Wi-Fi PSK Cracking

Using your lab book, complete the exercise, *Wi-Fi PSK cracking*.

This page intentionally left blank.

Course Roadmap

Introduction to IoT Network Traffic and Web Services

Exploiting IoT Hardware Interfaces and Analyzing Firmware

Exploiting Wireless IoT: Wi-Fi, BLE, Zigbee, LoRa, and SDR

SECTION 3

1. **Wi-Fi**
   Exercise: Wi-Fi PSK Cracking
2. **Bluetooth Low Energy**
   Exercise: BLE Device Interaction
3. **Zigbee**
   Exercise: Zigbee Traffic Capture
4. **LoRa**
5. **SDR**
   Exercise: Conducting a Replay Attack on IoT

This page intentionally left blank.

Introduction

How it works
Device discovery
Traffic capture and interaction
Authentication and encryption
Exploitation and device manipulation

Introduction

In this module, we will be working with Bluetooth Low Energy. Bluetooth has experienced a number of iterations over the years and Bluetooth Low Energy (or BLE) is the one making significant inroads into IoT, due to its significantly lowered energy consumption requirements.

We'll discuss the BLE stack, and the pairing process.

From there, we will get into how to perform traffic capture and interaction, and the hardware and software tools needed.

We will move into authentication and encryption, and discuss how the pairing process works, and BLE encryption when available.

Finally, we will wrap up with exploitation and device manipulation, and culminate in a fun lab to test your skills.

Bluetooth Low Energy (BLE)

BLE is a dramatic departure from the original Bluetooth Classic specification.

- Originally a 'fork' of BT Classic by Nokia

Focused on low power, low data rate transmissions

- BT Classic EDR – 2.1 Mbps
- BLE (4.x) – 1 Mbps

Battery life goal of 5-10 years

Marketing and technical standards provided by the Bluetooth SIG

Enormous adoption in most verticals, and near-ubiquitous in smartphones

Bluetooth Low Energy (BLE)

Many individuals look at BLE as a natural enhancement of Bluetooth Classic, when in fact, while it was originally based off BT Classic, it was 'forked' and modified significantly by Nokia. Nokia introduced the technology as 'Wibree' in 2006, but it was integrated back into the Bluetooth standard in 2010 as Bluetooth Smart.

Since then, BLE has enjoyed an explosion of adoption, being present on wearable devices, home automation, and almost every mobile phone on the market.

BLE is designed to come in at an extremely reasonable cost ($1 for the radio itself) and have a battery life of 5-10 years (depending on Bluetooth version).

Unlike Wi-Fi, one organization – the Bluetooth Special Interest Group (SIG) – manages the technical standard and the marketing and interoperability elements.

BLE Communications

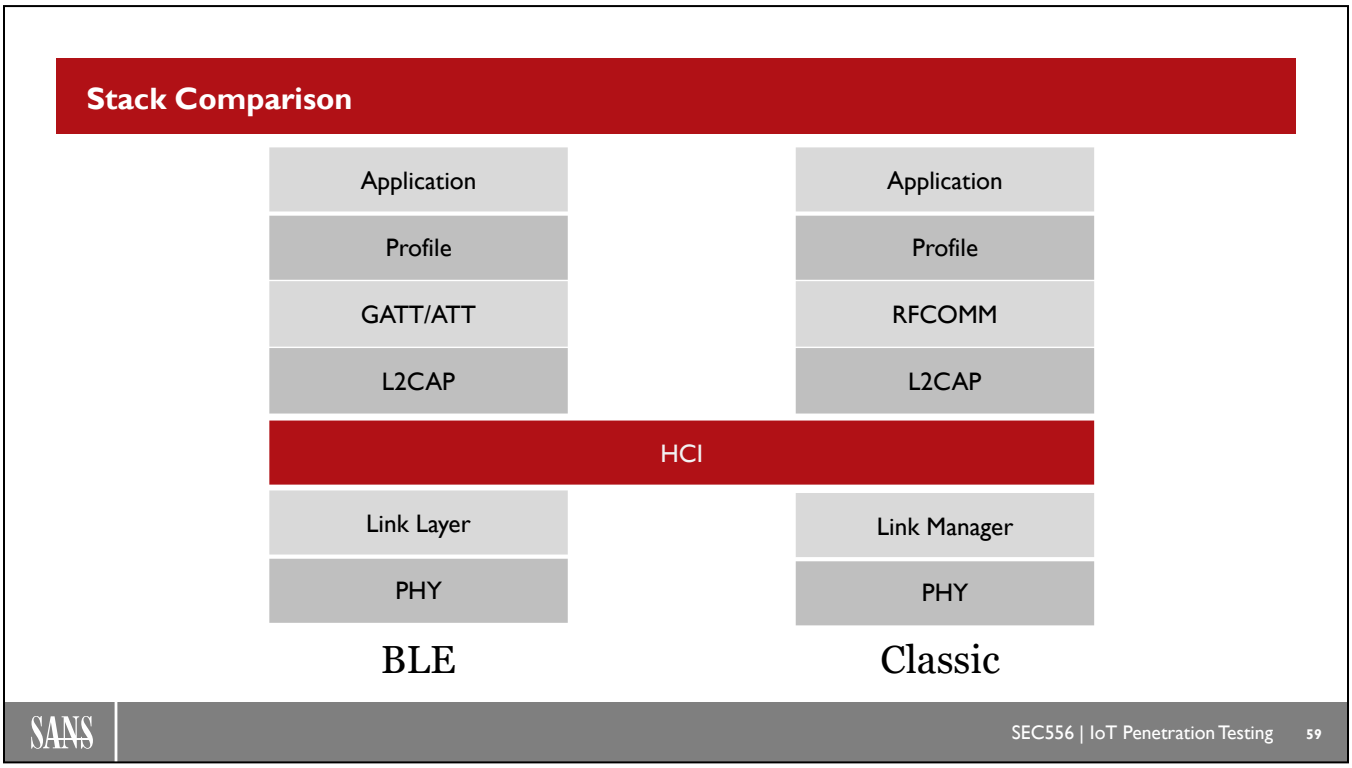
Many BT Classic 'like' elements, refined for power consumption and efficiency.

	BT Classic	BLE
Frequency Range	2.4GHz band	2.4GHz band
Range	~30M (Class 2)	~30M
Channels	79	40
Hop pattern	Erratic (algorithmically determined)	Predictable (mod 37 of current channel #)
Pairing	Derives LTK	Ephemeral (renegotiated per connection)
Data Exchange	Per profile methods	GATT/ATT

BLE Communications

BLEs implementation, compared to BT Classic's, is often streamlined while achieving similar performance goals with less power.

- Both BLE and BT Classic utilize the same 2.4GHz band, and therefore can utilize the same antenna if they coexist on the same hardware.
- BLE can achieve transmission ranges comparable with the most common types of BT Classic adapters.
- BLE uses 40 channels, 37 for transmissions and 3 for advertising, as opposed to BT Classics 79 total channels.
- While the hop pattern for BT Classic is erratic and determined algorithmically based on the Master device MAC address, the hop pattern for BLE is standardized and predictable.
- Pairing in BLE is more ephemeral than BT Classic and does not retain keys for reconnection with the other device. However, a different concept called *bonding* in BLE is introduced that will retain keys over time.
- Finally, while the exchange of data over BT Classic is siloed and dependent on the particular service and profile, the data in BLE is exposed via a protocol called GATT and accessed in a consistent fashion.



Stack Comparison

In comparing the overall stacks of BLE and Classic, we can see that on the surface, most layers appear the same. However, there are some changes, as we have reviewed.

We do have a significant restriction in the Host Controller Interface, or HCI. The HCI is an abstraction layer which prevents the user or application from interacting directly with lower-level functions. Features like pairing, bonding, and radio communications are not directly accessible.

The most obvious change is the replacement of the nonstandard RFCOMM layer with the standardized GATT/ATT implementation in BLE.

Practical Impact

BLE advertisements are easy to identify with a small number of channels.

BLE communications are easier to follow during the frequency hopping process.

Standardized data access protocols make it easy(!) to connect to and interact with BLE devices.

BLE devices are much easier to find and interact with than BT Classic…
for both authorized and unauthorized parties

Practical Impact

BLE devices are easier to find, follow, and connect to than BT Classic devices. This is great for users and applications, but not as much for security, as our adversaries can leverage this ease of use to their advantage.

Discovering BLE Devices

Discovery and enumeration of BLE devices is easy – if you have a BLE radio, you can do this too.

- hcitool
- bettercap
- nRF Connect (mobile)

Discovering BLE Devices

Discovering BLE devices is pretty straightforward, as long as you have a BLE radio. We will look at some common options for desktop and mobile.

Hcitool

Hcitool can be used to configure your BLE adapter, as well as scan for devices.

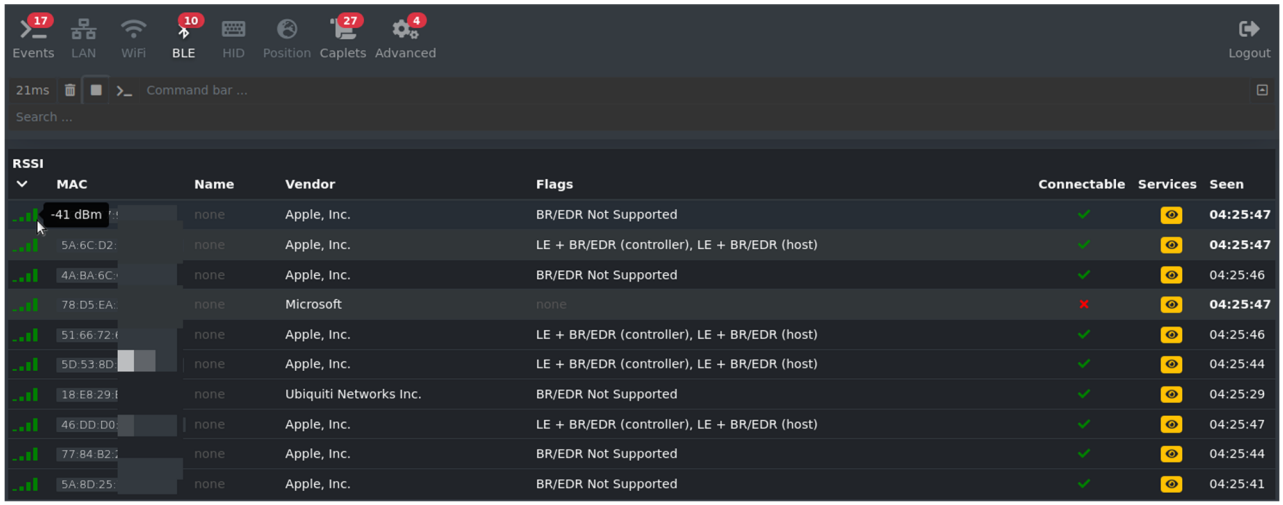
```
$ sudo btmgmt le on
hci0 Set Low Energy complete, settings: powered bondable ssp br/edr le
secure-conn

$ sudo hcitool lescan
F4:B8:5E:48:4A:F4 (unknown)
F4:B8:5E:48:4A:F4 LightShow Strings
<trimmed for brevity>
00:07:80:C5:F2:BA (unknown)
00:07:80:C5:F2:BA Pavlok-F2BA
^C
```


Hcitool

Hcitool allows for multiple capabilities within one tool, many around adapter configuration and settings– we can also readily identify devices with 'hcitool lescan'. Note it is common to see two entries for a given device – the second entry is hcitool pulling the 'friendly name' of the device which it may not have been able to immediately discern.

Bettercap for BLE Scanning

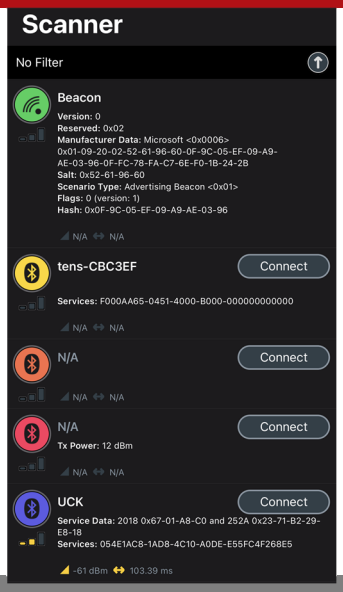


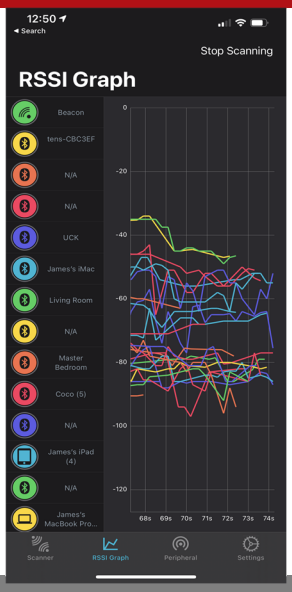
Bettercap for BLE scanning

Bettercap is a multi-purpose tool, and in this case, we are running the ble.recon module in bettercap. While this can all be viewed from the terminal, we have loaded the http-ui caplet to allow for easier GUI interaction. We can see that bettercap identifies signal strength (far left column), as well as MAC addresses, manufacturers (based on MAC OUI), supported Bluetooth types, and ability to scan for services (we will return to this functionality later).

Bettercap was developed by Simone 'evilsocket' Margaritelli and a community of contributors and is available at https://www.bettercap.org/.

Nordic nRF Connect





Nordic nRF Connect

Nordic nRF Connect is a multi-purpose tool from Nordic Semiconductor hat allows us to scan, connect, enumerate and bond with BLE devices. What is most exciting about nRF connect is that it is offered as a mobile app for both iOS and Android, which means you can scan and interact with your BLE surroundings, anywhere, at any time, in a fairly inconspicuous manner. We will return to interaction with the devices; for now, the screenshot on the left is the output of an active scan (note the 'connect' option on the devices) and a signal strength graph over time on the right.

Traffic Capture

Two types of traffic capture in BLE

- Below the HCI
- Above the HCI (GATT/ATT interactions)

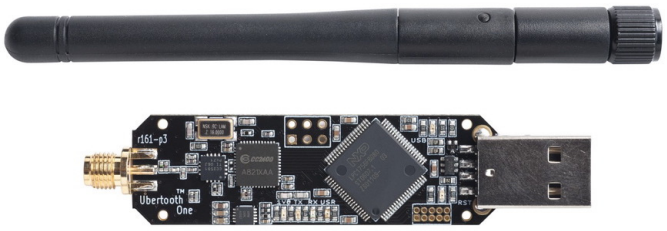
Traffic Capture

When we talk about the idea of traffic capture in BLE, this can mean a few things. It can refer to the low-level communications below the HCI (Host Controller Interface), which is not accessible from your typical BLE adapter –we need specially designed equipment for this.

Above the HCI interactions can also be captured and evaluated in tools like Wireshark. This can be accomplished with a number of different pieces of hardware.

Traffic Capture – Below the HCI

- Limited options here
- Need a radio that was purpose built to break the rules
- Enter the Ubertooth One.
- $120, custom hw/sw
- Custom sw *Ubertooth-btle* allows us to follow/capture traffic.



Traffic Capture – Below the HCI

When we talk about the idea of traffic capture in BLE, this can mean a few things. It can refer to the low-level communications below the HCI (Host Controller Interface), which is not accessible from your typical BLE adapter –we need specially designed equipment for this.

The Ubertooth One, created by the team at Great Scott Gadgets, allows for capture of the lower layer communications to be reviewed, used for PIN cracking, or passed to tools such as Wireshark.

The Ubertooth One still is undergoing active development and can be obtained from a number of online sources.

Traffic Capture – Ubertooth

```
$ ubertooth-btle -p -f

systime=1441512979 freq=2440 addr=8d651b4d delta_t=3.599 ms
86 9e d1 00 65 92 86 01 5d 3e 0e 5e 65 e0 61 9a 7d f7 04 c8 9f f5 45 00 ce f5 cc c8 8f
67 02 f5 4f a7 f5
Data / AA 8d651b4d (valid) / 30 bytes
    Channel Index: 17
    LLID: 2 / LL Data PDU / L2CAP start
    NESN: 1 SN: 0 MD: 0
    Data: d1 00 65 92 86 01 5d 3e 0e 5e 65 e0 61 9a 7d f7 04 c8 9f f5 45 00 ce f5 cc c8
8f 67 02 f5
    CRC: 4f a7 f5

$ ubertooth-btle -p -f -c capture.pcap
```


Traffic Capture – Ubertooth

In the above example, we are using the *Ubertooth-btle* to capture traffic. The Ubertooth One will watch for advertising devices, follow the channel hop pattern, and capture this output to a libpcap compatible file.

If the MAC address of a specific device of interest is known, we can follow that specific device by updating the switches to -t <BD_ADDR> , there <BD_ADDR> is the MAC address of the device of interest.

Traffic Capture – Above the HCI - NRF 51822

Adafruit BLE Sniffer/Friend, based on the NRF51822

- Two versions, only one is sniff capable
- Firmware updatable on nonsniffer, requires special hardware

Readily available, $25

- Amazon
- Adafruit

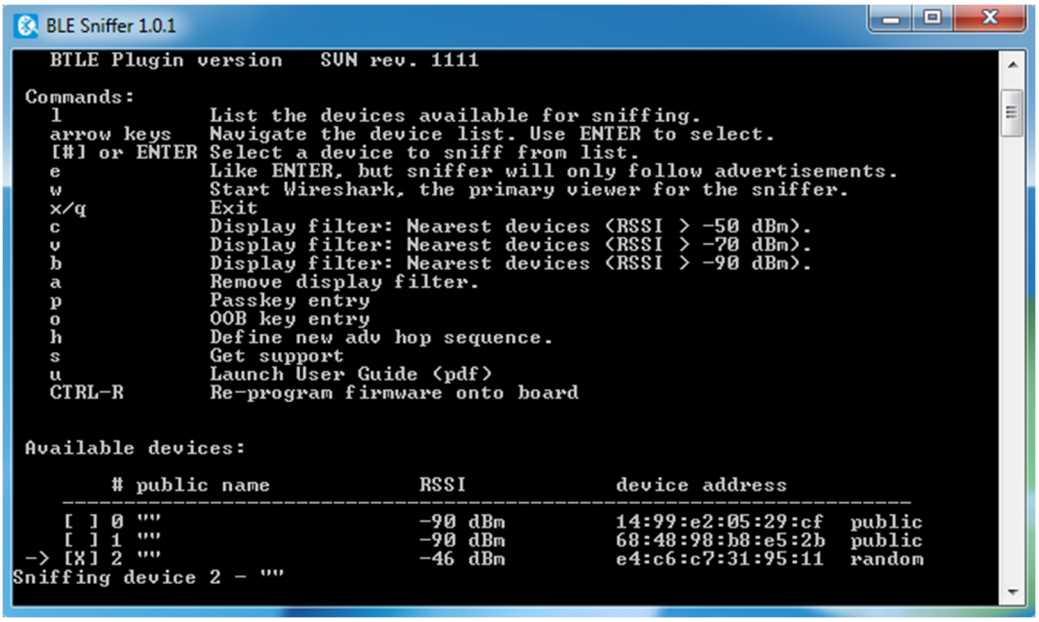
Works with Wireshark

- Windows and macOS app to pcap
- Adafruit Python libraries to pcap

Traffic Capture – Above the HCI - NRF 51822

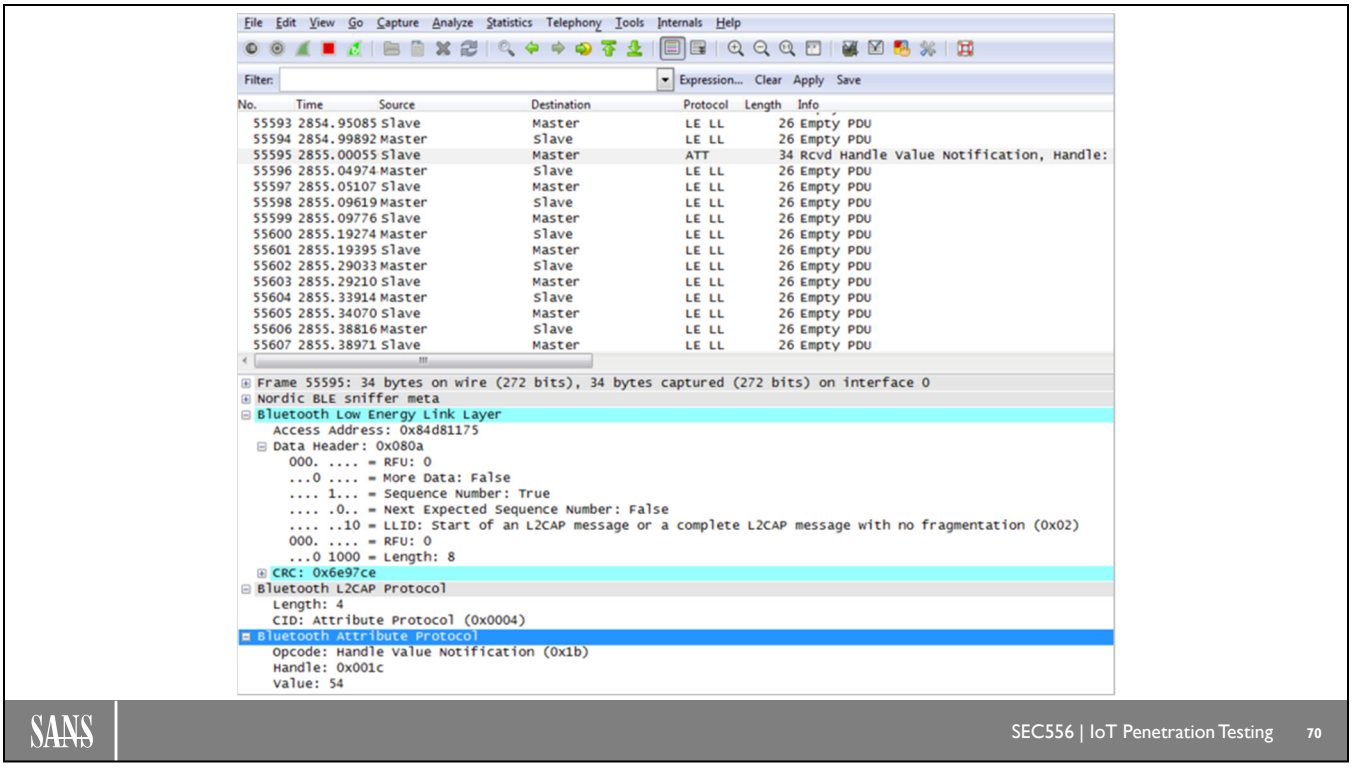
Another solution for sniffing is with the Adafruit BLE Friend, based on the Nordic NRF51822 chipset. While the NRF51822 is intended to be used as a BLE implementation for use in hardware, the team at Adafruit industries has implemented it in a USB dongle for use as a BLE development platform and as a sniffer for troubleshooting development. There are two versions of the Adafruit BLE Friend available: One with stock firmware to support application development and another specifically programmed with the sniffer firmware. While it is possible to transform the development version into the sniffer version by updating the firmware, changing back requires some additional, single-use programming hardware, the SEGGER J-Link JTAG debugger at $399. With that in mind, the nominal additional cost of the preprogrammed sniffer device is worth it and so is picking up a dedicated development model strictly for that purpose.

The preprogrammed sniffer devices, preconfigured for sniffing, are readily available through both Adafruit and Amazon for around $25, while the UART-based development version is $20.



BLE Sniffer Example

More details on the NRF sniffer for the BLE Friend can be found at https://learn.adafruit.com/introducing-the-adafruit-bluefruit-le-sniffer/nordic-nrfsniffer.



BLE Sniffer Output in Wireshark

More details on the NRF sniffer for the BLE Friend can be found at https://learn.adafruit.com/introducing-the-adafruit-bluefruit-le-sniffer/nordic-nrfsniffer.

BLE Authentication and Encryption

BLE Authentication is usually performed by pairing.
Pairing is a temporary connection between two devices in BLE:

- Phase 1: Pairing request
- Phase 2: Exchange TK/DH Key Exchange (pre/post BLE 4.2)

Longer term connection can be provided by bonding

- Phase 3: (bonding only) additional key exchange.

BLE Authentication and Encryption

The pairing process for BLE (4.0, 4.1) aka LE Legacy Pairing, uses a custom key exchange protocol. As part of the pairing process, devices exchange a Temporary Key (TK) and use it to generate a Short-Term Key (STK). This STK is subsequently used to encrypt the connection. The security of this process depends on the pairing method used to exchange the TK. The pairing process is performed in a series of phases, as described below.

Phase one: The initiator device sends a pairing request to the new device. The two devices exchange capabilities, authentication methods, the maximum link key size, and any bonding requirements. In this phase, both devices are exchanging capability information in order to determine the connection security in later phases. In phase one, all tata is on plaintext.

Phase two: When phase one is complete, the devices generate and/or exchange the TK. After the TK exchange, the devices exchange Confirm and Rand nonces to validate identical TK in use on both devices. Once the TK is confirmed to be identical on both devices, the TK and two nonces are used to create the STK. The STK generated is then used as input to AES-CCM to encrypt the connection.

Phase three: In this optional phase, transport-specific keys are exchanged for bonding, if those requirements were requested in phase one.

BLE 4.2 devices are backward compatible with 4.0 and 4.1 devices. BLE 4.2 is also capable of creating LE secure connections. Instead of a TK and STK, LE secure connections use a single Long-Term Key (LTK) for connection encryption. The LE secure connections LTK is generated using Elliptic Curve Diffie Hellman (ECDH) public key cryptography, ultimately with more robustness than the mechanism in 4.0 and 4.1. In LE secure connections, both phase one and phase three are exactly the same as they are in LE legacy connections. The only differences occur during phase two of the pairing process, with the change to the ECDH exchange.

BLE Authentication and Encryption Cont'd.

For ease of use, many BLE devices use a TK of 0 aka 'Just Works'.

- Yes. 0.

Other BLE TK Exchange options:

- Out of band pairing
- Passkey
- Numeric Comparison

However, the TK is generated, it is used to derive a Short-Term Key which encrypts with AES-CCM.

BLE Authentication and Encryption Cont'd.
We can exchange the TK in several methods:

Just Works:
In Just Works, the TK is set to 0. As a result, it's very easy for an attacker to use brute force for the STK and eavesdrop on the connection. Likewise, this method also offers no way of verifying the devices taking part in the connection, and it thus offers no MITM protection.

Out of Band (OOB) pairing:
With OOB pairing, the TK is exchanged using a different wireless technology, such as NFC. If the OOB channel is protected from MITM attacks, then it can be assumed that the BLE connection is also protected from MITM attacks. Of the three legacy pairing methods, OOB pairing is by far the most secure, provided that the OOB channel employs sufficient security methods.

Passkey:
The TK is a six-digit number that is passed between the devices and randomly selected by the user. One could generate a random six-digit number, display it on an LCD display, and have the user enter it into the other device with a HID.

Numeric comparison (4.2 only):
This pairing method follows the Just Works method but adds a final step. Once the BLE devices validate that the confirmation values match, both devices independently generate a final six-digit confirmation code created from both nonces. These six-digit values are displayed to the user on both devices. The user checks that both values match and verifies the connection, providing protection from MITM pairing attacks.

If an attacker is not listening in during the pairing process, the passkey method gives fairly good protection from passive eavesdropping. However, if an attacker is present during the pairing process and is able to sniff the values being exchanged, it is fairly trivial to brute force the TK and use it to derive the STK and decrypt the connection. To implement the highest level of security, one should implement OOB or the numeric comparison pairing method.

However, with BLE, the derived keys are not retained upon disconnect. Each reconnect derives a new set of temporary keys. Needless to say, if passkeys were required for each connection in order to derive keys potentially every few minutes, it would be terribly cumbersome and unusable. It would also be unusable in cases where BLE devices have no HID on the screen to display confirmation or interaction.

PIN Cracking

Because of the simplified hop sequence, PIN cracking is achievable with the right tools …

Need to be able to work below the HCI for this

- Ubertooth One is the most affordable alternative for this.

With observation of the pairing process, TK can be brute forced.

PIN Cracking

Due to the simplified hopping sequence compared to BT Classic, following a device through its frequency hops can be more readily accomplished, meaning the capability to follow a pair of devices during their pairing process. This in turn means that we can capture the complete pairing exchange. With this information, all that remains is to supply the one piece of information not provided over the air during the pairing – the PIN.

If the pairing exchange can be captured, we can pass this to the 'crackle' utility, created by Mike Ryan. The Ubertooth One is by far the least expensive tool for this task, and it has a utility 'ubertooth-btle' that facilitates this capture.

As the pairing process is designed to be fast (computationally inexpensive), brute forcing is typically accomplished very quickly. Crackle will also take encrypted packets and decrypt them as well.

TK Recovery

```
$ ubertooth-btle -p -f -c capture.pcap
^C
$ crackle -I capture.pcap -o decrypted.pcap

TK found: 000000
ding ding ding, using a TK of 0! Just Cracks(tm)
LTK found: 5f61c953f104a6bbe68b1d896a2f5d91
Done, processed 997 total packets, decrypted 17
```

TK = 0 = Just Works

The initial capture must observe a pairing exchange in order to recover TK.

TK Recovery

In this example of TK recovery after capturing a pairing request with ubertooth-btle, crackle was able to recover the TK of 0, meaning that the default key was used for Just Works.

Device Manipulation

Some of the most interesting things we can do with BLE can be performed with basic equipment – even our phones!

The simplicity and ease of access of GATT/ATT, and the ease of connecting to BLE devices, mean that almost anyone can do so…

What then?

Device Manipulation

BLE devices are designed to be easy to connect to and interact with. Think about Just Works pairing. We also have the GATT/ATT protocols to simplify access and interaction with a device by making all the calls to a device work the same way, regardless of service.

So, does that mean that we can connect to most BLE devices in the wild? Yes.

Does that mean we can manipulate values (where appropriate) and see what happens? Also, yes.

What will the result be? That depends on the device, and the implementation. In some cases, no changes of importance will be noted. In others though, we may be able to perform some meaningful actions with, or to, the device.

BLE Profiles

Every profile has its own data useful to the device.

Each profile exposes data in a hierarchical fashion.

GATT server handles data and descriptors.

- Basic element is a service, multiple services
- Each service has one or more characteristics
- Each characteristic has a descriptor

Every characteristic has several possible features.

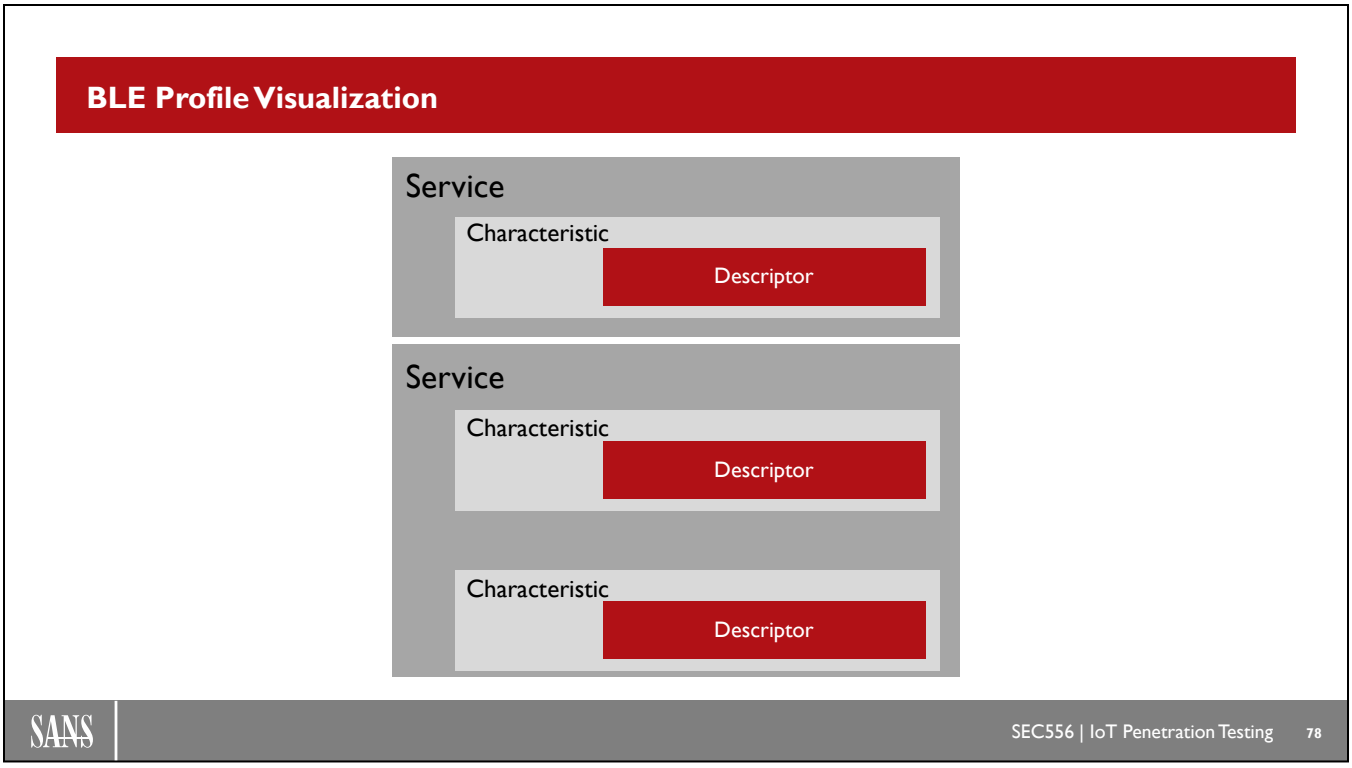
- Read, write, read/write, read/notify
- Notify acts as a subscription

BLE Profiles

A profile is identified by the type of data it represents, and hundreds of profiles are defined by the Bluetooth SIG. The Bluetooth SIG also recognizes that it may have missed some or that there is a need to have user-defined options.

Each profile has a hierarchical fashion as presented by the GATT server. The GATT server gives us access to services, service characteristics, and characteristic descriptors.

Every characteristic has several capabilities, some of which we may argue are security related. One can describe each data descriptor with the ability to read, write, and notify in various combinations. When examining descriptors, "notify" will perform a "push" to acknowledge any changes without a re-query.



BLE Profile Visualization

For those of us that learn with visuals, we can see the relationship between services, characteristics, and descriptors.

hcitool/gatttool

Tools with the BlueZ application suite

hcitool for scanning for available devices (previously seen)

gatttool for interaction with devices

- Discovery of services
- Query of service values
- Modification of writable values

Possible to script for service fuzzing

hcitool/gatttool

One of the most ubiquitous software suites for Bluetooth under Linux is the BlueZ suite. The BlueZ suite features many utilities, some of which work for both BLE and Classic. From the BlueZ suite, the combination of hcitool and gatttool form a powerful combination for interacting with BLE devices.

We first start with hcitool with the lescan verb in order to discover advertising devices. This will automatically select the first Bluetooth device present in the system. We may need to specify an interface, if multiple ones are present, with the -i hciX option (where X is the preferred Bluetooth adapter). The output will reveal a scrolling list of advertising devices, revealing their BD_ADDRs. If the device's friendly name is available, hcitool will attempt to resolve it as well.

Once the BD_ADDRs have been recovered from the advertising devices, it is possible to connect to them with Just Works, if available. We can connect using gatttool and begin interacting with the profiles. We can attach to a specific device using the -b flag and specify the BD_ADDR of the device to connect to. Additionally, specifying the –I option will place gatttool in interactive mode. Once in interactive mode, we can interact with the profiles by setting the primary profile and then exploring the various components, standard or otherwise.

Instead of using interactive mode, we can specify all of the same profile options at the command line. By doing so, we can use some scripting to increment the values sent to each component, creating a basic, scripted fuzzer.

hcitool/gatttool Example

```
$ hcitool lescan
F4:B8:5E:48:4A:F4 (unknown)
F4:B8:5E:48:4A:F4 LightShow Strings
<trimmed for brevity>
00:07:80:C5:F2:BA (unknown)
00:07:80:C5:F2:BA Pavlok-F2BA
^C

$ gatttool -b 00:07:80:C5:F2:BA –I
[   ][00:07:80:C5:F2:BA][LE]> connect
[CON][00:07:80:C5:F2:BA][LE]> primary
attr handle: 0x0001, end grp handle: 0x0007 uuid: 00001800-0000-1000-8000-
00805f9b34fb
<trimmed for brevity>
[CON][00:07:80:C5:F2:BA][LE]> char-desc 0x0007 0xffff
handle: 0x000b, uuid: 2901
<trimmed for brevity>
[CON][00:07:80:C5:F2:BA][LE]> char-read-hnd 0x000b
Characteristic value/descriptor: 4d 61 6e 75 66 61 63 74
[CON][00:07:80:C5:F2:BA][LE]> char-write-req 0x000b 00
Characteristic Write Request failed: ... <trimmed>
```

THINK ABOUT

While gatttool service identifiers are accurate, they are not intuitive. They must be manually mapped to the list of documented services provided by the Bluetooth SIG.

hcitool/gatttool Example

Our initial interaction with hcitool is to perform a scan of advertising BLE devices. Once discovered, the BD_ADDRs can be used with gatttool in interactive mode to discover primary services and to read and write with the various characteristics.

Bettercap for Device Interaction (1)

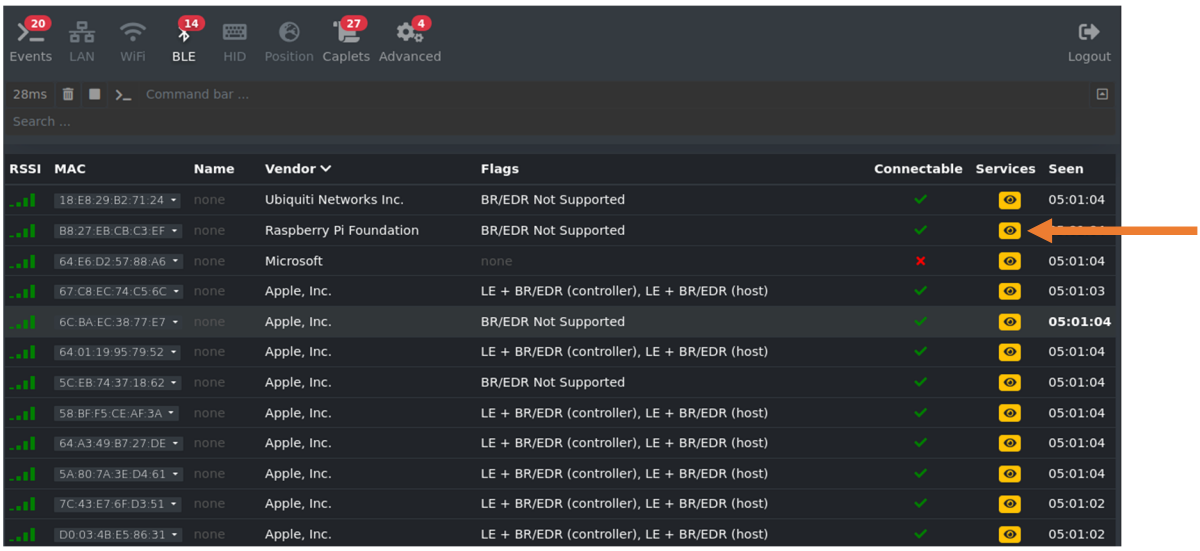
Bettercap allows for:

- Connection to discovered BLE devices.
- Enumeration of services/characteristics/descriptors.
- Identification of features of characteristics.
- Modifications of descriptors when possible.

Bettercap for Device Interaction (1)

We discussed bettercap earlier in the context of scanning for BLE devices. However, bettercap also does an amazing job of connecting to a device and identifying services, characteristics, descriptors, and the features (permissions) of those characteristics. While gatttool exploration is somewhat manual, bettercap goes a long way towards simplifying the process.

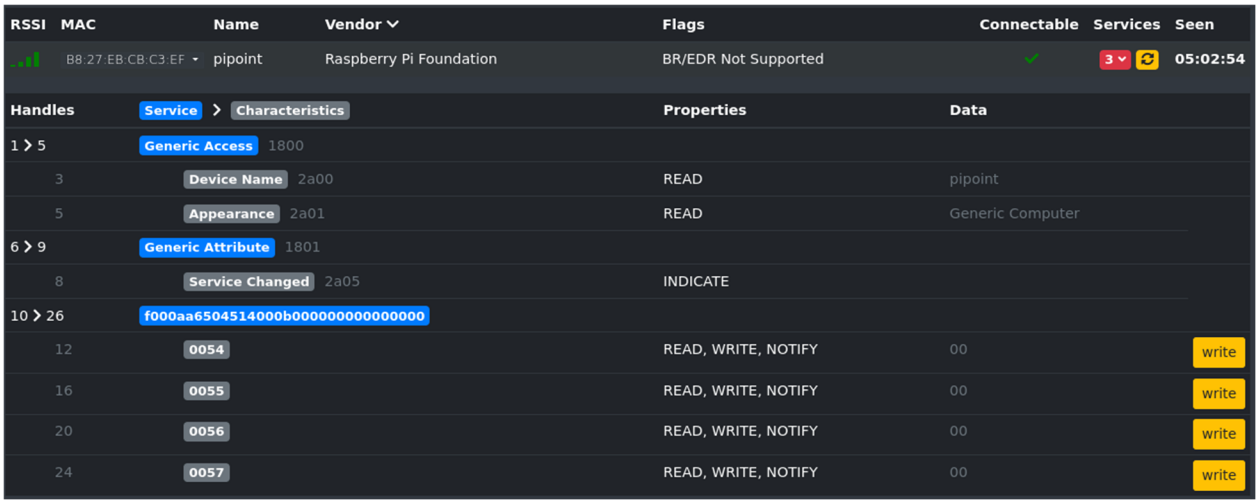
Bettercap for Device Interaction (2)



Bettercap for Device Interaction (2)

Here's a view of bettercap performing a scan. In this case, we can click on the 'eye'-con over services to connect to the device and attempt enumeration. Once this is complete, bettercap will allow us to drill into the results.

Bettercap for Service Enumeration



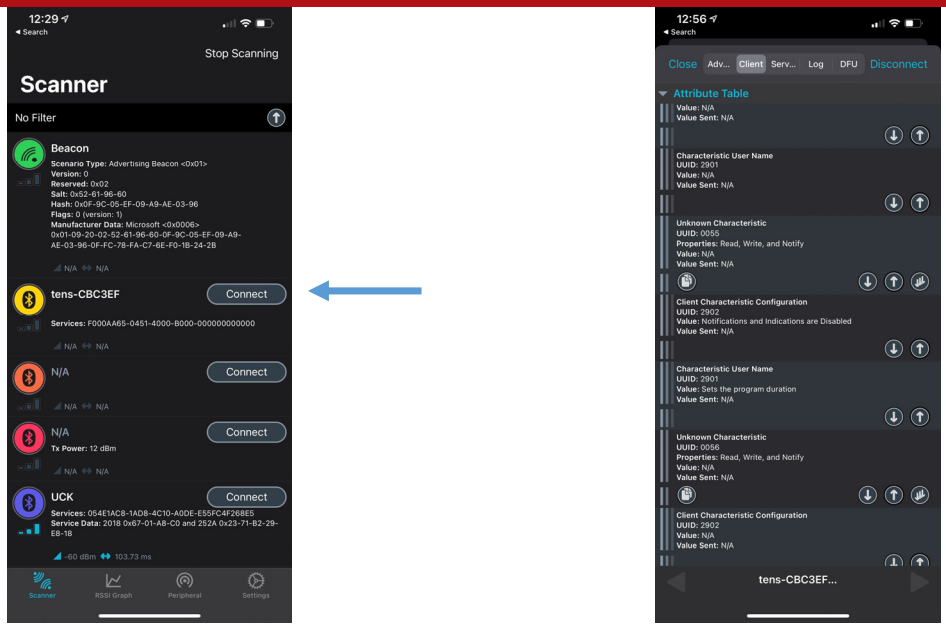
RSSI	MAC	Name	Vendor	Flags	Connectable	Services	Seen
	B8:27:EB:CB:C3:EF	pipoint	Raspberry Pi Foundation	BR/EDR Not Supported	✓	3	05:02:54

Handles	Service > Characteristics	Properties	Data	
1 > 5	Generic Access 1800			
3	Device Name 2a00	READ	pipoint	
5	Appearance 2a01	READ	Generic Computer	
6 > 9	Generic Attribute 1801			
8	Service Changed 2a05	INDICATE		
10 > 26	f000aa6504514000b000000000000000			
12	0054	READ, WRITE, NOTIFY	00	write
16	0055	READ, WRITE, NOTIFY	00	write
20	0056	READ, WRITE, NOTIFY	00	write
24	0057	READ, WRITE, NOTIFY	00	write

Bettercap for Service Enumeration

Here, we can see bettercap's handiwork. Bettercap has been able to pull common characteristics like the Device Name and Appearance, and decode the values associated with those. In this case the UUID associated with Device Name and Appearance mapped to well known identifiers maintained by the Bluetooth SIG, which bettercap understands. We see other characteristics as well, that we can write to. However, they are not identified as 'well-known' and so we cannot discern their purpose. The only way to find out is to experiment...with permission, of course!

Nordic nRF Connect for Device Interaction



84

Nordic nRF Connect for Device Interaction

Nordic nRF Connect wields one distinct advantage over some of the other offerings we've seen here – it can run from a mobile device, easily concealed.

We saw nRF Connect before for scanning. Now, if we connect to a device, we can open the 'client' tab at the top and see all the services and characteristics. As with bettercap, nRF will map well known UUIDs, and show features of these characteristics as well.

Summary

BLE took several steps to simplify the protocol and make it easier to use.

These changes also simplify a path to attack of these devices as well.

In many cases, 'interesting' results can be yielded by GATT interactions with almost any BLE radio.

With BLEs enormous market penetration, it will continue to remain a valuable target for our adversaries.

Summary

In this module, we explained the essentials of BLE, including pairing/bonding, architecture, and practical attacks against the device as well as pairing processes.

We discussed bettercap earlier in the context of scanning for BLE devices. However, bettercap also does an amazing job of connecting to a device and identifying services, characteristics, descriptors, and the features (permissions) of those characteristics. While gatttool exploration is somewhat manual, bettercap goes a long way towards simplifying the process.

BLE is expected to remain a target due to its extreme market share and ease of access.

BLE should be employed in most cases after performing threat modeling, and if BLE is not needed for a particular device, it should be disabled.

Exercise: BLE Device Interaction

Using your lab book, complete the exercise, *BLE Device Interaction*.

This page intentionally left blank.

Course Roadmap

Introduction to IoT Network Traffic and Web Services

Exploiting IoT Hardware Interfaces and Analyzing Firmware

Exploiting Wireless IoT: Wi-Fi, BLE, Zigbee, LoRa, and SDR

SECTION 3

1. **Wi-Fi**
   Exercise: Wi-Fi PSK Cracking
2. **Bluetooth Low Energy**
   Exercise: BLE Device Interaction
3. **Zigbee**
   Exercise: Zigbee Traffic Capture
4. **LoRa**
5. **SDR**
   Exercise: Conducting a Replay Attack on IoT

This page intentionally left blank.

What Is Zigbee?

Low-power, low–data rate wireless protocol
- Built on IEEE 802.15.4 specification

250 Kbps maximum throughput

Small, lightweight software stack

Mesh or star topology support

5-year goal design goal for modest battery

Range commonly 10–100 meters
- Mesh topology can extend distance

What Is Zigbee?

Zigbee is a low-power, low–data rate wireless protocol designed on top of the IEEE 802.15.4 specification. Supported by the Zigbee Alliance, Zigbee is used to provide connectivity between low-power devices, providing a standards-based mechanism for embedded device communication and data exchange.

As a low-power protocol, Zigbee supports a maximum data rate of 250 Kbps with a common range between 10 and 100 meters. Integrated into the specification is support for both mesh and star networking topologies using a small, lightweight stack requiring approximately 120 KB of storage for the base specification. Designed to operate on a single set of batteries for five years, Zigbee has found entry into several vertical markets where prior wireless technologies were not suitable for the desired applications.

Why Do Attackers Care about Zigbee?

Zigbee controls the kinetic world more than any other wireless packet protocol.

Manipulating the physical world through wireless introduces new risks.

Many of the past mistakes are repeated, again.

Used across many verticals, big and small manufacturers

Manipulating Zigbee affects the physical world in many ways now and in the foreseeable future.

Why Do Attackers Care about Zigbee?

Zigbee is an increasingly attractive technology for attackers as it has been implemented in many areas where other wireless technologies were not a suitable fit, often monitoring or controlling devices that interface with the kinetic world. With the ability to manipulate devices in the physical world over Zigbee, attackers have new opportunities to attack and exploit critical systems leveraging Zigbee for control interfaces. Combined with several vulnerabilities in Zigbee repeated from mistakes in earlier protocols, Zigbee offers opportunities to an attacker that are seldom possible through previous wireless protocols, making it an attractive target now and for the foreseeable future.

Wide Ranging Adoption

Smart thermostats, electric/gas meters, smart energy

ICS, IIoT

Lighting, Audio/Visual

Physical access control, alarms

Adaptation of older, non wireless connected technology

Wide Ranging Adoption

Next, let's examine several examples of Zigbee devices and deployments. A common Zigbee device that will be arriving in many households throughout North America and the world are smart thermostats. These thermostats interface with metering devices from local electric and gas meters over Zigbee, collecting and reporting real-time pricing information and billing information. From the thermostat, the consumer and the utility gain access to control home machinery, commonly including heating and air conditioning systems, with future control over other appliances in the home, such as water heaters, dishwashers, washing machines, and dryers.

Through the ability to collect real-time energy pricing information, consumers can decide how and when to heat or cool their homes and when to use other appliances, depending on the cost of utility resources. For example, if the cost of electricity is high at 6:00 p.m. in the evening, consumers can decide to defer running a dishwasher until later, when the cost of electricity drops following peak demand times. Electric utilities can design programs where the utility can remotely turn off appliances during peak utilization periods that threaten the stability of the electrical grid.

The Siemens APOGEE product line includes networked Field Level Controllers (FLCs) used to interface with industrial heaters, exhaust fans, air conditioning units, lighting, and other industrial building machinery. The APOGEE products represent an opportunity for building managers to control and monitor equipment within a facility that otherwise lacks network connectivity through the generic FLC interfacing mechanism.

Jay Hendrix, Siemens Manager for Wireless Solutions, is quoted in a press release regarding the APOGEE Zigbee devices indicating that "the network can't be compromised because the signal is automatically able to circumvent obstructions and find its target." Many businesses will find the APOGEE product line valuable as a mechanism to modernize existing machinery to accommodate remote control and monitoring services.

While smart thermostats are much anticipated as a welcome tool for energy conservation, they represent an opportunity for an attacker to implement similar controls over appliances.

The Kwikset SmartCode systems represent new technology for home security systems, allowing a user to unlock a door using a supported mobile device either over the internet or by entering a preallocated code on the numeric pad on the lock. The administrator can allocate multiple authorized PINs for unlocking doors, leveraging time-of-day authorization privileges, or even gaining one-time access codes to grant access to contractors or other in-home service providers.

The door lock system interfaces with a Zigbee appliance inside the home, connecting to the internet over other wireless or wired connections. Through this Zigbee integration, the door lock can be integrated with other Zigbee peripherals in the home, such as smoke detectors designed to alert the fire department in the event of a fire, triggering an unlock event to grant access within the home to firefighters.

History of Exploitation

IoT Goes Nuclear: Creating a ZigBee Chain Reaction

- Ronen, O'Flynn, Shamir, Weingarten surmise the development of a Zigbee worm, Zigbee wardriving, and attack delivery via drone

Subsequent Phillips hue chained attacks

- Zigbee to internal network pivot

Smart energy, AMI

- Treated as critical infrastructure, DoS, stack corruption ==Denial of Power, Theft of Power and Disruption of Grid

History of Exploitation
Over the years Zigbee has seen a multitude of attacks. While many are unpublished and relatively minor in scope, several have been identified as being quite devastating on a larger scale.

In 2016 Ronen, O'Flynn, Shamir, and Weingarten discuss the possibilities of delivering a wormable Zigbee exploit after discovery via ZigBee wardriving and delivery to a facility via drone. They later demonstrated drone delivery of a worm against a cooperative research partner (https://www.theverge.com/2016/11/3/13507126/iot-drone-hack). In subsequent demonstrations, the wormable attacks were used to be more targeted, and they turned them into some chained attacks against the Phillips Hue ecosystem, creating the ability for an attacker to gain a foothold in the network throught a connected Hue bridge, allowing for eventual pivot into internal networks via the internet. We'll discuss this attack more in depth shortly.

For Smart Energy deployments, some of the largest implementations have been noted to be vulnerable, resulting in denial-of-service conditions. Several computer security research teams have released testing tools (Termimeter, Optiguard and others) involving ZigBee or adjacent technologies with the capabilities for large scale meter disconnect/reconnect events. These attacks could turn more devastating, with eventual attacks leading to theft of power, and more concerning, a large disruption in distribution by triggering large scale disconnect/reconnect events.

Phillips Hue Chained Attacks

Inject spoofed frames to change color of a bulb

Exploits bulb to upload backdoored firmware

- Acts as unresponsive to the app
- User resets bulb to reset and re-add to network

On re-add bulb forges Zigbee frames to Hue bridge

- Large quantities of packets cause a heap-based overflow on the bridge

Overflow allows for backdoor install on bridge

- Bridge connects back to attacker, now as a pivot to the rest of the network

Phillips Hue Chained Attacks

In 2016-2017 a team of Eyal Ronen, Colin O'Flynn, Adi Shamir, Achi-Or Weingarten discovered some attacks and methodology in Zigbee implementations that would allow for the compromise of a network with an attached Zigbee to ethernet hub. They found that they could compromise a Phillips Hue bulb over the Zigbee network and cause it to behave erratically and re-flashes the firmware on the bulb, forcing a disconnect from the network.

Once re-flashed, the user notes the erratic behavior and re-joins the bulb to the network, adding the compromised bulb to the correct Zigbee network. Once the bulb can participate the attacker can then instruct the compromised bulb to craft large quantities of Zigbee frames destined for the Zigbee to ethernet bridge causing a heap-based overflow on the bridge. With the overflow triggered the attacker can then implant a backdoor on the bridge, causing a connection to the attacker's command and control infrastructure. The bridge can now be used as an ethernet connected pivot point inside of the affected network.

The teams' research can be found at https://eyalro.net/publication/rosw17.html.

Due to design limitations with the Phillips Hue infrastructure, Checkpoint confirmed that the attacks against the bulbs are still present in 2020, even after patches are applied. Their research can be found at https://blog.checkpoint.com/2020/02/05/the-dark-side-of-smart-lighting-check-point-research-shows-how-business-and-home-networks-can-be-hacked-from-a-lightbulb/.

Zigbee Background

Initially released in 2004
- Evolution based on additional security features

Most operation in the 2.4 GHz band
- Also, at 900 MHz (North America), 850 MHz (Europe) where extra distance is required

16 channels at 5 MHz wide
- Channel 11=2.405 GHz to channel 26=2.480 GHz

Maximum frame size is 127 bytes

Zigbee Background

Now that we've seen several examples of Zigbee technology and deployments, let's examine some of the technology and details supporting the system. Zigbee technology was initially released in 2004 with little fanfare and market acceptance, lacking any security or additional profile features. In 2006, an amendment to the 2004 specification was released, introducing security features including encryption and message integrity support. Immediately following that, the Zigbee Alliance began to publish additional profile specifications, highlighting the areas where Zigbee is attractive for embedded device interconnectivity.

Despite the added security features included in Zigbee-2006, several vulnerabilities were recognized in how key management and allocation were performed. To address these issues and to support the growing need for high-security Zigbee deployments, the Zigbee Alliance released two versions of two additional amendments to the Zigbee-2006 specification. The Zigbee-2007 specification added support for two profiles. The first, known simply as *Zigbee-2007*, was designed to support home and business automation profiles. The second, known as *Zigbee-PRO*, included additional device authentication and key derivation functions with the concept of a Zigbee Trust Center (TC) responsible for authenticating the identity of other Zigbee devices joining the network.

Zigbee commonly operates in the 2.4 GHz band worldwide, though in February 2010, the Zigbee Alliance also began to start certifying sub–1 GHz devices at 900 MHz in North America and 850 MHz in Europe. Zigbee uses the Direct Sequence Spread Spectrum (DSSS) coding, similar to that of 802.11b networks. In the 2.4 GHz band, Zigbee offers 16 channels starting at Channel 11 (2.405 GHz) and ending at Channel 26 (2.480 GHz). Each channel is 5 MHz, with the frequency representing the center point of the channel (e.g., Channel 11 spans 2.4025 GHz to 2.4075 GHz).

Zigbee has a maximum frame size of 127 bytes, which is limiting for many applications. In Zigbee-2007, support was added for payload fragmentation over a maximum of 256 packets.

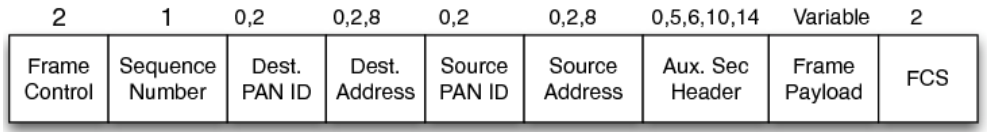
MAC Layer

Max payload 114 bytes
- Dependent on 2 or 8-byte network addressing

Four frame types: Beacon, data, command, ACK

Addresses are 2 or 8 bytes

PAN ID is like a BSSID
- Support for multiple networks on the same channel



MAC Layer
The MAC layer specifies the frame formatting operation, addressing formats and additional options for upper-layer security. With a maximum payload size of 114 bytes, a transmitter can embed a little data in each packet, also making buffer and heap overflow exploits and general fuzzing attacks challenging.

The MAC layer supports four types of frames:

- Beacon: Beacon frames are used by the coordinator to advertise the availability of a Zigbee network and to provide timing synchronization services.
- Data: Data frames convey data messages to other Zigbee devices.
- Command: Command frames are used to describe a specific action on the Zigbee network, such as an association request or a disassociation notice.
- ACK: ACK frames are used to confirm the successful receipt of a packet without conveying any additional data.

When joining the Zigbee network, a device is allocated an address at the MAC layer that is unique within the Zigbee PAN. The administrator can choose to use a 16-bit or a 64-bit address, using the longer address form if the network will include more than 65,535 hosts. Within a given PAN, the PAN ID address (selected by the coordinator when the network is formed) is used to identify the network to which the traffic belongs, differentiating multiple Zigbee networks in the same physical space. A Zigbee router can bridge traffic between two PANs through the use of the source PAN ID and the destination PAN ID FCS.

The MAC layer defines several fields, many of which can be zero bytes in length depending on earlier characteristics in the frame:

- Frame Control: The Frame Control field defines eight additional subfields (and three reserved bits) that include the frame type, security-enabled bit, ACK request, frame delivery mode, frame version, and source and destination addressing modes (16-bit or 64-bit).
- Sequence Number: The Sequence Number is a module 256 counter incrementing by one for each frame sent by a single source.

- Destination PAN ID: The Destination PAN ID is used to indicate the PAN to which the frame is headed. When inter-PAN bridging is not in use, this field identifies the PAN to which the traffic belongs (similar to an 802.11 BSSID).
- Destination Address: This is the 16-bit or 64-bit address of the remote Zigbee device receiving traffic.
- Source PAN ID: The Source PAN ID is used to specify a different PAN identifier to use for bridging the frame from the current PAN and is in use only when bridging between PANs is in use.
- Source Address: This field contains the 16-bit or 64-bit address of the local Zigbee device sending traffic.
- Auxiliary Security Header: The Auxiliary Security Header is used when the NWK layer leverages security to specify the types of keys used and the level of authentication and encryption used for the network.
- Frame Payload: A variable amount of data may be present in the Frame Payload.
- FCS: This field contains a 16-bit frame check sequence based on an unauthenticated CRC.

CCM* Protocol

Variation of AES-CCM
- 128-bit key length

Network key: Shared among all devices, most common key used

Link key: Unique for two devices

CCM* Protocol

The Zigbee specification leverages the CCM* (Counter Mode, Cipher Block Chaining Message Authenticity Check) protocol for encryption and integrity protection and is a variation of the NIST-approved CCM protocol. CCM leverages the AES cipher as both an encryption method (using Counter Mode) and a hashing algorithm to provide a message integrity check (MIC, using Cipher Block Chaining Mode) with the same key. CCM is also used for IEEE 802.11-2007 networks as part of WPA2 wireless deployments.

The variation adopted by the Zigbee Alliance (the asterisk in CCM*) is to accommodate various options for MIC lengths, including 16-bit, 32-bit, and 64-bit, as well as an option for no MIC, with the 32-bit MIC being the default. CCM* also accommodates the use of a MIC without any encryption, an unsupported method in the standard CCM protocol.

Leveraging the CCM* protocol, traffic is encrypted and/or authenticated using one of two key types:

- Network Key: Introduced in Zigbee-2006, the most common security method for Zigbee is to use a single key, known as the *Network Key*, shared across all devices in the network.
- Link Key: An optional mechanism in Zigbee-PRO, the Link Key can be used to establish a unique key between two communicating devices in a Zigbee network.

Unlike the MIC methods used in 802.11 networks, the Zigbee MIC can be used to cover the entire frame contents, preventing an attacker from replaying frames from one destination to another.

Key Provisioning

Preinstallation, devices have key set at factory (master or network key).

Key transport, key is sent in plaintext when device joins network.

SKKE negotiation, requires pre-shared master key.

- Derives unique link keys between device pairs (similar: 4-way handshake)

OTA key distribution is insecure and challenging with Zigbee and keys are often unprotected in the hardware.

Key Provisioning

Delegating keys to devices in the provisioning process is one of the challenging components in Zigbee networks. Due to the lack of multifunction operating systems with alternate network connectivity (such as wired ports) and a limited human interface devices (HID), provisioning, rotating, and revoking keys is largely an insecure or manual process. Up to and including the Zigbee-PRO specification, three key provisioning options have been available:

- Preinstallation: Used in Zigbee-2006, Zigbee-2007, and Zigbee-PRO, the preinstallation method allocates a master or network key when a device is provisioned at the factory (or reprovisioned in the field by a programmer). With preinstalled keys, the provisioning process can be secured while performed in the manufacturing facility, but it lacks automatic key rotation or revocation.

- Key transport: Used in Zigbee-2006, Zigbee-2007, and Zigbee-PRO, the key transport method issues the key in plaintext over the network to the device when it joins the PAN, after authenticating the device through a MAC address ACL. The key transport method allows for the simple revocation and issuance of new master or network keys but exposes the integrity of the key by transferring it over the air, unprotected.

- SKKE negotiation: Used in Zigbee-PRO, the Symmetric-Key Key Establishment (SKKE) negotiation method is used to derive a unique link key between two devices based on common knowledge of a master key. While the master key must initially be shared with the two devices through preinstallation or over-the-air key transport, the SKKE negotiation accommodates secure key derivation for both parties, with the opportunity to renegotiate keys at subsequent connections. The SKKE negotiation method is similar to the operation used in the IEEE 802.11-2007 4-way handshake.

Upper-Layer Zigbee Security

Zigbee profiles can define an additional layer of security.

Smart Energy Profile

- Leverages ECC for authentication, encryption using PKI

Healthcare, Remote Control, Home Automation, Building Automation

Not all profiles are created equal.

Smart Energy Profile: v1 with ECC, v2 with faulty TLS implementation

Upper-Layer Zigbee Security

In addition to the security features defined in the Zigbee specification, Zigbee application profiles can include an additional layer of security for environments where that is warranted. For example, Smart Energy Profile (SEP) version 1.0 defined by the Zigbee Alliance leverages Elliptic Curve Cryptography (ECC) for authentication and encryption based on a well-structured public key infrastructure (PKI). While this adds a new layer of security to Zigbee devices, ECC is also resource intensive and costly to implement due to the licensing requirements from Certicom, the company holding patents surrounding the use of ECC. At the time of this writing, SEP 2.0, which discards the use of ECC in favor of patent-unencumbered technology (TLS), has been ratified as of 2014. SEP 2.0 is not without issues, specifically the TLS implementation lacks a certificate revocation list; if a SEP 2.0 device is stolen/lost there is no ability to revoke the certificate, thus preventing access to the network.

Other Zigbee profiles using the Healthcare, Remote Control, Home Automation, and Building Automation specifications leverage the standard Zigbee security mechanisms. At the time of this writing, the Zigbee Telecom and Zigbee Retail profiles are also in development and have yet to announce their intentions for security adoption.

KillerBee

Framework and tools for exploiting Zigbee networks

Simplifies capture and injecting traffic, packet decoding and manipulation

Tools designed to aid in recon, exploitation

Pure Python, even Python3!

Ongoing development

- Originally developed by Joshua Wright
- Current development by RiverLoop Security

KillerBee

KillerBee is a framework for Linux systems written by this author, designed to provide an interface for exploiting Zigbee and IEEE 802.15.4 networks. With KillerBee, analysts and developers have access to simple sniffing and packet injection tools and APIs as well as facilities to decode and manipulate packet captures. Using KillerBee, the reconnaissance and exploitation of Zigbee and IEEE 802.15.4 networks is straightforward through several tools included in the suite.

As a pure Python interface, KillerBee integrates well with other Python tools, including Sulley and Scapy, for advanced Zigbee testing. The author has sought to require minimal library dependencies to aid in the quick and simple setup of KillerBee. KillerBee is made available under a BSD license, allowing the software to be used, free of charge, for commercial and noncommercial projects. KillerBee is available at https://github.com/riverloopsec/killerbee.

KillerBee Hardware

Texas Instruments CC2531 System-on-Chip ($10)
- Pick up two devices for sniff + inject

Varieties available with on-board antenna, RP-SMA

Available from Amazon, eBay

Stock firmware sniff capable

Custom Bumblebee firmware required for Tx
- Needs downloader cable ($5) and CCdebugger ($16)
- Lousy cost for one time use, but firmware a moving target
- Also, possible to flash with RaspberryPi, Arduino

NOTE

At time of writing, KillerBee and the Bumblebee CC2531 firmware are to be considered "alpha" and may be subject to issues in transmitting Zigbee packets.

KillerBee Hardware

Over the years, we have seen several hardware interfaces for use with KillerBee., some of which had been discontinued by the manufacturer, or requires a significant investment in programming hardware to make them useful. Currently, one of our best options is the API Mote v4, buy River Loop Security, made by the same folks that currently maintain the KillerBee suite of tools. With a $150 cost per device, there is a little bit of a barrier to entry, especially if one wants to sniff and receive at the same time.

In recent months, a new hardware interface is supported by KillerBee, the Texas Instruments CC2532, available from popular electronics sites including Digikey, Amazon and eBay for around $10. When ordering hardware, this author recommends purchasing two devices for simultaneous receive and transmit capabilities. The hardware itself is very simple, consisting of a multi-purpose radio, a USB microcontroller, and a printed circuit board (PCB) antenna.

The stock devices are capable of sniffing but need to be flashed with the custom "Bumblebee" firmware for transmit or replay operations. Unfortunately to flash the Bumblebee firmware an external programmer must be used, the additional investment of the programmer ($6) and cables($16) is lousy, in that you only need it for a few moments. Currently the firmware and support with KillerBee is "alpha" so obtaining the programming hardware may make sense, should updates to the firmware be made to enhance stability. It has also been documented that flashing is possible with a Raspberry Pi, or an Arduino, but the process is a bit more convoluted. This may be a reasonable alternative, if you already have a Raspberry Pi or Arduino in your arsenal.

KillerBee Arsenal

zbid – List available devices supported
zbdump – "tcpdump -w" clone
zbconvert – Convert capture file formats
zbreplay – Replay attack
zbdsniff – OTA crypto key sniffer
zbgoodfind – recover keys from hardware

KillerBee Arsenal

In addition to providing a development API, KillerBee includes several tools designed to explore and attack Zigbee and IEEE 802.15.4 networks:

- zbid: Lists the available RZUSB interfaces on the local system.
- zbdump: A "tcpdump -w" clone supporting both libpcap and the proprietary Daintree Sensor Network Analyzer (SNA) packet capture file formats. (Daintree SNA is no longer commercially available as a Zigbee protocol analyzer.)
- zbconvert: Converts packet captures from Daintree SNA to libpcap and vice versa.
- zbreplay: Replays traffic from a specified packet capture file.
- zbdsniff: An over-the-air crypto key sniffer.
- zbfind: A GUI for locating and tracking Zigbee devices.
- zbgoodfind: Searches a memory dump or other binary file for the presence of a network key.
- zbassocflood: A proof-of-concept tool to flood the association table of a target Zigbee Router or Coordinator.
- zbstumbler: Actively scans for Zigbee network devices.

Several of the tools are named after other similarly purposed tools from other authors in homage to their great work.

```
$ sudo zbid
Dev     Product String  Serial Number
3:9     CC2531 USB Dongle     None
$ sudo zbstumbler
zbstumbler: Transmitting and receiving on interface ‘3:9'
New Network: PANID 0x4EC5  Source 0x0000
        Ext PANID: 39:32:97:90:d2:38:df:B9
        Stack Profile: Zigbee Enterprise
        Stack Version: Zigbee 2006/2007
        Channel: 15
New Network: PANID 0x858D  Source 0x0000
        Ext PANID: 00:00:00:00:00:00:00:00
        Stack Profile: Zigbee Standard
        Stack Version: Zigbee 2006/2007
        Channel: 11
^C
202 packets transmitted, 183 responses.
$ sudo zbdump -i 3:9 -f 11 -w out.dump
Warning: You are using pyUSB 1.x, support is in beta.
zbdump: listening on 'CC2531 USB Dongle', channel 11, page 0 (2405.0 MHz), link-type
DLT_IEEE802_15_4, capture size 127 bytes
zbdump: listening on ‘3:9', link-type DLT_IEEE802_15_4, capture size 127 bytes
```


KillerBee Demo

On this slide, we look at examples of KillerBee in use where we identify the available interfaces on the system with zbid, followed by an active device discovery using zbstumbler. Zbstumbler identifies two PANs, disclosing the source address of the device reporting the presence of the PAN; the PANID; and additional information, including the extended PANID, Zigbee stack profile, and version and channel numbers. You can stop zbstumbler by issuing a CTRL+C interrupt.

Finally, a packet capture is retrieved using zbdump on the ‘3:9' interface on Channel 11, saving the contents to the file out.dump.

KillerBee Attack Example 1

Zigbee keys can be pre-installed or over-the-air (OTA) provisioned.

Most preinstallation methods require re-flashing device to change the key.

- Not optimal with limited-flash-write devices

OTA key delivery allows for frequent key rotation.

Key sent in plaintext

Attacker observes a new device connecting to the network, identifies encryption key

KillerBee Attack Example 1

Next, we'll look at several Zigbee attack examples and how they can be implemented using the KillerBee framework. In this example we examine observing key distribution methods for Zigbee: flashed from the factory with default keys or updated by the end user, Over the air with Key Transport method in plaintext at the time of network join, or in an encrypted fashion over the air with SKKE (requiring pre-shared key).

The over-the-air (OTA) provisioning option. This method is advantageous, as it accommodates frequent key rotation among devices and avoids reflashing Zigbee nodes each time the key is rotated (a valuable proposition when working with limited-flash-write lifetime devices).

The weakness in OTA key provisioning is that the key is initially delivered to the node in plaintext, allowing an attacker to observe the key through traffic sniffing.

zbdsniff

OTA crypto key sniffer

Reads from libpcap or Daintree SNA files automatically

- Easily searches all captures for OTA crypto keys

Wrap it around find+xargs to search all packet captures, as shown below.

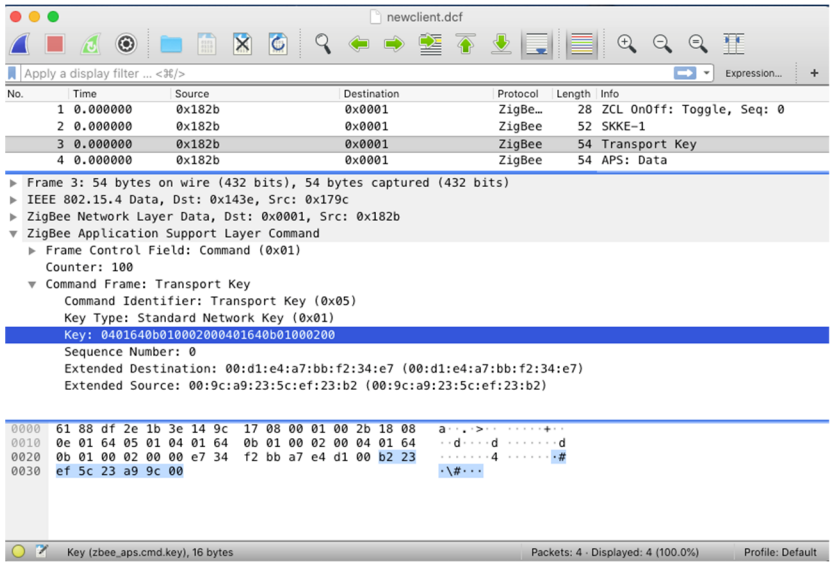
```
$ find . \( -name \*.dcf -o -name \*.dump \) -print0 | xargs -0 zbdsniff
Processing ./ct80-rapidsedesk.dump
Processing ./stumbler-chan15.dcf
Processing ./newclient.dump
NETWORK KEY FOUND: 00:02:00:01:0b:64:01:04:00:02:00:01:0b:64:01:04
  Destination MAC Address: 00:d1:e4:a7:bb:f2:34:e7
  Source MAC Address:      00:9c:a9:23:5c:ef:23:b2
Processing ./ct80-conn3.dcf
```


zbdsniff

Zbdsniff implements the OTA crypto key sniffing attack, reading from libpcap or Daintree SNA packet capture files. When an OTA key exchange is identified, zbdsniff displays the key information, including the source and destination addresses of the devices involved.

To search through multiple packet capture files with zbdsniff, we can wrap the tool in a "find" command, piping the output to `xargs` and the `zbdsniff` command, as shown.

Keys in Wireshark



Keys in Wireshark

In this example of keys noted with zbdsniff, we can also observe them in Wireshark. As an alternative to zbdsniff we can use the *zbee_aps.cmd.key* filter in Wireshark to discover OTA keys as well.

zbreplay

Straightforward, unintelligent replay attack
The analyst decides what to replay and observes the response.
Daintree DCF files are ASCII, easy to chop up

```
$ sudo zbreplay -i 3:10 -f 20 -r out.dump
Warning: You are using pyUSB 1.x, support is in beta.
zbreplay: retransmitting frames from 'out.dump' on interface 'CC2531 USB Dongle' with a
delay of 1.0 seconds.
True
13 packets transmitted
```


zbreplay

The zbreplay tool implements a simple replay attack, taking a packet capture as an input and replaying the contents of the traffic with a specified interframe delay interval (with a default of one second between each frame). The analyst decided what traffic to replay, specifying the packet to transmit in the packet capture file. Daintree DCF files are particularly attractive for this attack, since the attacker can easily carve out packet extracts and manipulate the contents of the ASCII-based DCF capture file format.

KillerBee Attack Example 2

Zigbee networks have lots of small, distributed devices.

Encryption key is a resident in RAM.

Find a device you'd like to "audit".

Leverage device to retrieve encryption key from hardware.

Use recovered key to access network or decrypt all previously captured traffic.

KillerBee Attack Example 2

The nature of Zigbee and IEEE 802.15.4 devices is to have a large number of small, distributed devices communicating. Especially when deployed in outdoor environments, these devices are prime targets for theft, creating new opportunities for an attacker.

In many networks, the network key is stored in flash until the device boots, where it is moved to RAM and then used to encrypt and decrypt traffic on the network. If an attacker is able to capture the contents of flash or RAM, they can use this information to search for the encryption key by repeatedly attempting to decrypt a valid, encrypted packet.

This attack is not entirely dissimilar to WPA2-PSK networks, where each client device has knowledge of the network key.

Key Recovery: zbgoodfind

Obtain firmware image via JTAG or SPI/I2C.

Convert hex file to binary file with objcopy.

Using an encrypted packet, attempt to decrypt with each possible key value in RAM

Use zbgoodfind to apply firmware as decryption key to captured packets, one by one offset

```
$ objcopy -I ihex -O binary chipcon-2430-mem.hex chipcon-2430-mem.bin
$ zbgoodfind -R encdata.dcf -f chipcon-2430-mem.bin
zbgoodfind: searching the contents of chipcon-2430-mem.hex for encryption keys with the
first encrypted packet in encdata.dcf.
Key found after 6397 guesses:  c0 c1 c2 c3 c4 c5 c6 c7 c8 c9 ca cb cc cd ce cf
```


Key Recovery: zbgoodfind

Once the GoodFET has extracted the contents of the RAM of a target Zigbee or IEEE 802.15.4 device, we can use the KillerBee zbgoodfind tool to search the dump file for the presence of the network key. This is performed by using a packet encrypted with the network key and attempting to decrypt the packet with all the possible keys in the RAM dump file. Since RAM on embedded devices is usually small, this is a short-duration attack.

In this example, we use the goodfet.cc (ChipCon) tool to dump the contents of RAM from a target device, saving the contents to the file chipcon-2430-mem.hex. This .hex file is an ASCII file that we convert to a pure binary memory dump using the objcopy command, as shown. Finally, we use zbgoodfind to read the encrypted packet capture ("-R encdata.dcf") and the binary RAM dumpfile ("-f chipcon-2430-mem.bin") and start using brute force for the possible key values. After 6397 guesses, zbgoodfind reports success and displays the key used to encrypt the packet capture file.

Note that zbgoodfind can be used to brute force the encryption key from any given binary file. Another common use case for zbgoodfind is to eavesdrop on all the activity between a microprocessor and the radio interface performing encryption and decryption with a bus sniffing tool, such as the TotalPhase Beagle I2C/SPI Protocol Analyzer (https://www.totalphase.com/products/beagle-i2cspi/) and pass the bus data contents as a binary file to zbgoodfind to search through.

Summary

Zigbee is a growing low-power wireless protocol.

Rapidly gaining market acceptance and deployment numbers

KillerBee is a hacker-friendly interface for experimenting with Zigbee security.

Zigbee interfaces with the kinetic world, controlling insignificant and critical devices.

Summary

Zigbee is a growing, low-power wireless protocol that is rapidly gaining market acceptance and greater deployment numbers. Deployed in many critical areas, the Zigbee protocol is well suited for the interconnectivity of embedded devices. The protocol is not without its risks, however, and to date, several attacks have been demonstrated against Zigbee networks using commercial and open-source tools, including KillerBee.

In order to encourage the growth of low-power, low–data rate wireless protocols targeting embedded devices such as Zigbee, we need additional research and analysis of the security of these systems. In this fashion, we can gain an understanding of the risks we introduce with this technology and make informed decisions as to the benefits versus the costs of these deployments.

We concluded with a brief look at Z-Wave technology. Z-Wave is a growing wireless technology popular in consumer networks and likely destined for additional growth through the continued market adoption of the Internet of Things (IoT). KillerZee is an early suite of attack tools and an API for developing additional tools, using readily available and off-the-shelf radio components with the RFcat firmware.

Exercise: ZigBee Traffic Capture

Using your lab book, complete the exercise, *Zigbee Traffic Capture*.

This page intentionally left blank.

Course Roadmap

Introduction to IoT Network Traffic and Web Services

Exploiting IoT Hardware Interfaces and Analyzing Firmware

Exploiting Wireless IoT: Wi-Fi, BLE, Zigbee, LoRa, and SDR

SECTION 3

1. **Wi-Fi**
   Exercise: Wi-Fi PSK Cracking
2. **Bluetooth Low Energy**
   Exercise: BLE Device Interaction
3. **Zigbee**
   Exercise: Zigbee Traffic Capture
4. **LoRa**
5. **SDR**
   Exercise: Conducting a Replay Attack on IoT

This page intentionally left blank.

What Is LoRa?

Low-power, long distance wireless protocol

- 2-3 Km coverage outdoor obstructed
- 5-7 Km in unobstructed line of sight

0.3kbps to 50kbps data rate

Minimal battery, up to 10 years

Developed by Cycleo, currently owned by Semtech

Lots of EU adoption, N. America on uptake

What Is LoRa?

LoRa is a low-power long distance wireless protocol. It is significantly different in its technology stack then ZigBee AND amazingly different in the distance it can cover on modest power – up to 10 years on a modest battery with aggressive power saving! Where ZigBee is typically 10-100 meters, LoRa can cover 2-3 kilometers in an obstructed, outdoor area and 5-7 kilometers with clear line of sight outdoors.

Depending on how many devices are connected to any given access point and channel, we can range between 0.3kbps (with lots of clients) and 50kbps (with 1 to few clients). LoRa is clearly a low bandwidth protocol.

LoRa was originally developed by Cycleo and transferred to its current owner Semtech. During its course of adoption, the initial adoption was in Europe, with a massive uptake in adoption in North America, most recently with the additional enablement of LoRa in Amazon Sidewalk.

Why Do Attackers Care about LoRa?

Control of the kinetic world.

Observation (and interaction) over long distances

- No close proximity required.

1.3+ Million internet connected gateways, 178 million end nodes

Manipulating LoRa will increasingly be used to manipulate and monitor the physical world.

Why Do Attackers Care about LoRa?

LoRa is an increasingly attractive attack target, in that its increased adoption it being used for more and more control and monitoring of the physical world.

Because of the long distance of LoRa transmissions, we can observe, and ultimately interact from extreme distances. This makes observing an attacker sniffing your LoRa unlikely; they may be kilometers away! Similarly, those looking to interact with LoRa networks may go unobserved because of the distance.

At the time of this writing The Things Network, one of the largest LoRa providers claims more than 1.3 million gateways are installed worldwide with 178 million connected nodes. These numbers increase every day.

Uptake in Adoption

Parking meters, smart meters

Amazon Sidewalk

Animal tracking

- Black Rhinos and sea turtles and reindeer! (oh my!)

Water monitoring, irrigation

Natural disaster monitoring

Country/locale wide IoT networks

Uptake in Adoption

LoRa ha seen adoption across many different verticals. The Things Network has noted many interesting applications, as well as some noted from public sources.

Some of the larger implementations are intending to cover entire countries, or very large locales. One of the new ways we are seeing widespread coverage is with the recent enablement of Amazon Sidewalk, effectively turning modern Amazon IoT devices into a widespread LoRa network. On the more community or government driven networks we find that they are often used in support of community technology, such as parking meters, smart energy, and even collection of sensor data for water, wind and seismic activity.

Some commercial applications have been noted for monitoring crop irrigation and water levels. Some amazing application of LoRa features wildlife management for determining location, of endangered animals, such as Black Rhinos, sea turtles and even reindeer!

Where LoRa Excels

Distance, distance, distance

Compliments 5G, Bluetooth, Wi-Fi

- Distance, low power and network design

Network reliability

Security by default…

Where LoRa Excels

LoRa has a few areas in which it excels over other technologies, for a low power, low data rate protocol. The most significant feature that is above and beyond other technologies (given its design considerations) is distance. The several kilometer distance range, with a modest battery is above and beyond all similarly capable protocols. It does, however, compliment and augment 5G, Bluetooth and Wi-Fi, enhancing functionality, and integration with additional IoT systems.

LoRa networks, when connection points are deployed with appropriate density, maintain a high level of reliability, much like we see with appropriately designed cellular or enterprise Wi-Fi networks.

In a refreshing turn of events, LoRa was apparently designed from the start with well developed security in mind!

LoRa Background

Occupies ISM band
- N. America 915MHz, EU 433MHz & 868MHz APAC* 923MHz

Uses Forward Error correction to support long range

Class A (all): low-power end device
- Tx then Rx stimulus (all devices)

Class B (beacon): End device with deterministic latency
- Rx based on timeslots, then Tx (Ping slots)

Class C (continuous): low-latency end device
- Continuous Rx, with Tx responses

LoRa Background

Lora operates in the ISM bands, depending on where you are on the planet. In North America, we find most devices operate at 915MHz, with a few at 433MHz. Europe uses 433MHz and 868MHz, and the Asia Pacific region uses 923MHz with some variations into 865-857MHz as well.

In similar fashion to the advances with Bluetooth Low Energy 5 CodedLE, LoRa can achieve its long range partially due to the adoption of Forward Error Correction (FEC). When the two devices are far apart, the reception of signal can be degraded, making several bits of a transmission "unreadable". With the inclusion of FEC, the additional overhead allows for reconstruction of the bad bits, making the transmission whole and consumable again.

LoRa has the concept of device class:

Class A (all): low-power end device

Tx then waits for Rx stimulus (all devices) useful for impetrations such as environmental monitoring, fire detection, animal tracking. Typically, battery powered.

Class B (beacon): End device with deterministic latency

Rx based on timeslot scheduling, then Tx (Ping slots) useful for Smart Meter energy usage, temperature reporting . Also, often battery powered, but with reduced battery life.

Class C (continuous): low-latency end device

Continuous Rx, with Tx responses, always waiting for data. Useful for gateways, Utility meters. Typically powered by a continuous, non-battery, power source.

RF Challenges (1)

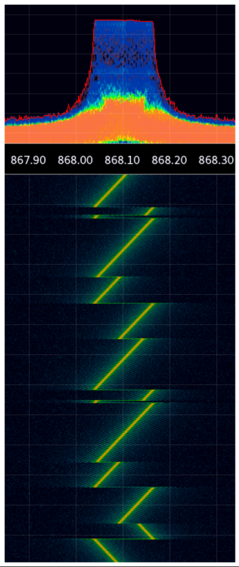
Proprietary Spread Spectrum modulation

- Chirp Spread Spectrum (CSS)
- Chirp signal that continuously varies in frequency
- Chirp frequency == bandwidth

Chirp pulses make it resistant to multipath, fading, and doppler effects

Significant technical details mired in secrecy

- Semtech guards implementation details to protect its interests, dominance



RF Challenges (1)

LoRa uses a proprietary spread spectrum modulation, a derivative of Chirp Spread Spectrum. CSS represents each bit of payload data with multiple chirps of data. The spread information send rate is known as the symbol rate. The ratio of symbol rate and chirp rate is the spreading factor (SF) which represents the number of symbols sent per bit.

LoRa can balance data rate for sensitivity by using a fixed channel bandwidth and the amount of spread between 7 and 12.

More chirps are sent at lower Spreading Factor, increasing the data rate. A higher Spreading Factor sends fewer chirps, decreasing the data rate. Data sent with a higher SF requires the radio to be on more/longer, consuming more power to the detriment of battery life. On the converse, with the radio more actively powered for the higher SF more samples be gathered to improve the sensitivity of the receiver. This helps LoRa be resistant to issues with multipath, signal fade and movement/doppler effects.

There is still much for us to learn about the RF implementation, and the CSS implementation. Semtech, in an effort to retain its intellectual property has chosen to keep many of the implementation details secret, only licensed to specific manufacturers.

RF Challenges (2)

64 uplink channels, 125Khz each

8 downlink channels, 500Khz each

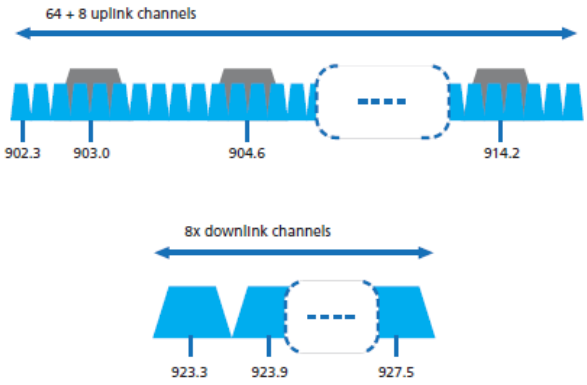
8 uplink channels, 500Khz each

Variable adjacent groups of chirp channels for a Spreading Factor

Spreading Factor determined by

- Bandwidth/data rate
- Uplink/downlink
- Estimated terrain/distance

Varies slightly with other locales not using 900MHz frequency range



RF Challenges (2)

In North America, there are 64, 125 kHz LoRa uplink channels defined, each at 125Khz wide. There are eight 500 kHz uplink channels as well as eight, 500 kHz downlink channels. In North America, gateways can have up to 64, 125 kHz uplink channels as well as eight 500 kHz uplink and downlink channels, typically found in "enterprise" APs or gateways.

The LoRa physical layer is intended for low throughput, low data rate, long-range applications. For a fixed channel bandwidth, the higher the spreading factor, the higher the processing gain, resulting in an increase in sensitivity and, therefore, an increase in distance, however, the time on air will also increase.

LoRa processing gain is introduced in the RF channel by multiplying the data signal with a spreading code or chip sequence. The total signal is spread over a wider range of frequencies, allowing the receiver discover a transmission with a signal-to-noise ratio (SNR). The amount of spreading applied to the original data signal is called the *spreading factor* (SF). LoRa has six defined spreading factors (SF7 to SF12). The larger the spreading factor used, the more distance the signal can travel and still be received before correction with FEC is needed.

LoRa MAC

Provides the basis for data exchange

No security, encryption or authentication

Low end devices will implement LoRa, but not LoRaWAN

Vulnerable to eavesdropping

LoRa MAC

The MAC layer for LoRa provides the basic access mechanism to the network, but no security is implemented at this layer. With the overall lack of security at this layer, LoRa PHY is vulnerable to sniff/eavesdropping with the appropriate hardware.

LoRa is purely a physical (PHY), or bits layer implementation. Instead of copper or fiber, the air is used as a medium for transporting LoRa radio waves from a transmitter in an IoT device to a receiver in a gateway, and vice versa. We will find that some simpler devices with transfer data using LoRa itself, but not the LoRaWAN layer.

LoRaWAN MAC Layer

Encryption available with AES-128 ECB

OTAA 1.0

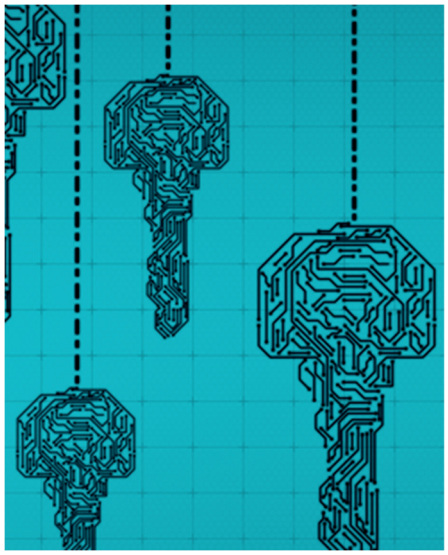
- Static fixed (App) keys used as input to TLS like join procedure, MIC

OTAA 1.1

- Same Join procedure as 1.0, adds separate static key for MIC key calculation
- Separate MIC keys for uplink and downlink

ABP

- No join procedure, static, fixed keys used directly



LoRaWAN MAC Layer

LoRa is the physical (PHY) layer, i.e., the wireless modulation used to create the long-range communication link. LoRaWAN is often the protocol that delivers secure bi-directional communication. Encryption is available at this layer Using AES-128 with electronic codebook mode encryption (ECB).

When a device joins the network AppSKey and a NwkSKey are generated. The NwkSKey is shared with the network, but the AppSKey is kept a secret. These session keys will be used for the duration of the session.

The NwkSKey is used for secure communication between the IoT device and the Network Server. This key is used to validate each message by a Message Integrity Check (MIC). The MIC is similar to a checksum, except that it prevents tampering through use of AES-CMAC The AppSKey) is used for encryption of the data payload. These two session keys (NwkSKey and AppSKey) are unique per device, per session. Additionally, an AppKey can be used to derive the two session keys during the activation procedure, using either a default AppKey, or a unique AppKey per device.

If dynamically activated with Over The Air Activation (OTAA), the keys are re-negotiated for every session, unless they are statically assigned, and not generated with OTAA.

LoRa features 3 different join methods for creation of keys. OTAA 1.0 uses static AppKeys for TLS like functionality. OTAA 2.0 improves on the 1.0 implementation by adding additional, separate keys for MIC calculation, one each for Tx and Rx. Finally, the simplest and easiest to implement is ABP, which uses static, fixed device addresses and user defined keys.

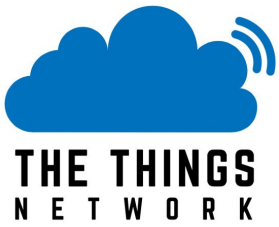
LoRaWAN Servers

Control various parts of participation in the network

- Gateway, Network, Application, Join, Identity

Application servers responsible for end processing of data

Public servers available through the Things Network, others



LoRaWan servers can be condensed to a limited number of devices with several *services* supporting the network.

LoRaWAN Servers

Application servers are responsible for securely handling, managing and interpreting sensor application data, and create application downlink payloads to the connected IoT devices.

The server types include:

- The gateway provides access from RF to a back-end TCP network.
- The LoRaWAN network server (LNS) manages the entire network, dynamically controls the network, and establishes secure 128-bit AES connections. The network server ensures the authenticity of every sensor on the network and the integrity of every message.
- Application servers receive and process application data.
- The Join server manages the over-the-air activation process for end devices to be added to the network.
- Identity servers provide access to databases that store applications with their end devices, gateways, users, organizations, OAuth clients and authentication providers. It also manages access control through memberships and API keys.

We can set up our own servers, or use large publicly available servers, such as those offered by the global Things Network, and other providers.

Upper-Layer LoRa Security

Unencrypted MQTT data

Possible to implement MQTT security

- Authentication and authorization
- SSL/TLS
- Payload encryption with mcrypt

Upper-Layer LoRa Security

Once we have established out connection to the network using LoRa and LoRaWAN, we can add additional layers of security on top of those layers with built in application layer security.

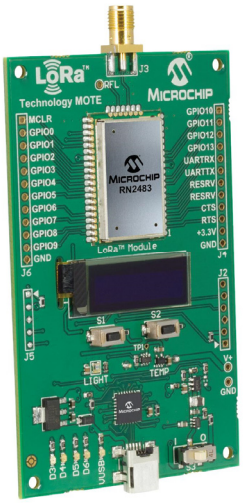
One common protocol in use over LoRa/LoRaWAN is MQTT lightweight messaging distribution and publishing. MQTT is often found using cleartext messaging; however, it is possible to further protect our LoRa/LoRaWAN/MQTT traffic, by implementing some basic authentication and authorization over MQTT, and encrypting the subsequent data with TLS for data in transit and mcrypt for payload encryption. The recommendations are to always encrypt MQTT with TLS 1.2 in transit. Depending on the processing cycles available on the IoT device, this may not always be possible, as a proper TLS implementation requires a fair amount of overhead for certificate generation, application and verification of validity and certificate chains.

LoRa Tools (1)

Dedicated hardware implementation
- Microchip RN2483
- RFM98/RFM95 (433/900 MHz)

"C" with the Arduino IDE
- Supporting dozens of platforms

Python



LoRa Tools (1)

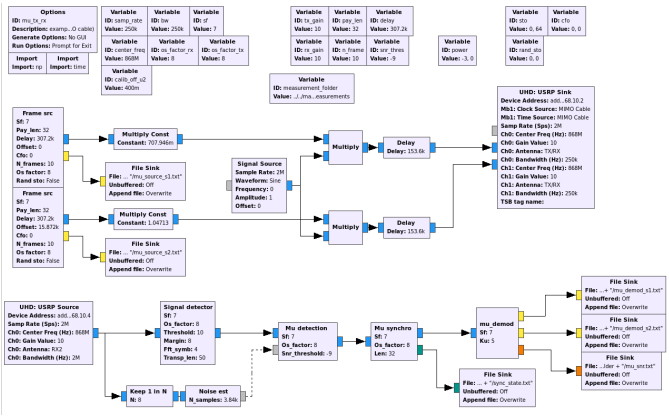
In order to begin interaction with LoRa networks, we need to use a supported LoRa chipset, which are only supplied by a limited number of manufacturers, keeping Semtech's secrets and dominance of the market in place. Two common chipsets are the Microchip RN2483 and the HopeRF RFM98 (433MHz) and the RFM95 (900MHz), all three of which are available across multiple LoRa development platforms and can be found in currently shipping IoT devices.

Interaction with these LoRa radios in the development side of the house (most applicable to our security research and interaction), can be done in a C-like language under the Arduino IDE, with appropriate libraries, supporting dozens of individual microcontroller platforms. In addition, there are several libraries for supporting direct development and use with python as well.

LoRa Tools (2)

Software Defined Radio
- RTL-SDR
- HackRF One, other TX capable platforms

Python and GNURadio
- Implements LoRaWAN security
- Just provide recovered keys!



LoRa Tools (2)

In addition to the purpose-built hardware, it is also possible to use the power of a software defined radio, such as the RTL-SDR, for receive only of LoRa packets. When a transmit capable SDR is used, such as the HackRF, it is possible to have full interaction with a LoRa network. The HackRF is not the only transmit capable platform for interaction, as any that are compatible with GnuRadio should work.

With a software defined radio, we can use the power of both python and GNURadio to interact with LoRa/LoRaWan networks, even including appropriate implementations of the security framework: all we need to do is provide the appropriate keys.

LoRa Attack Example 1

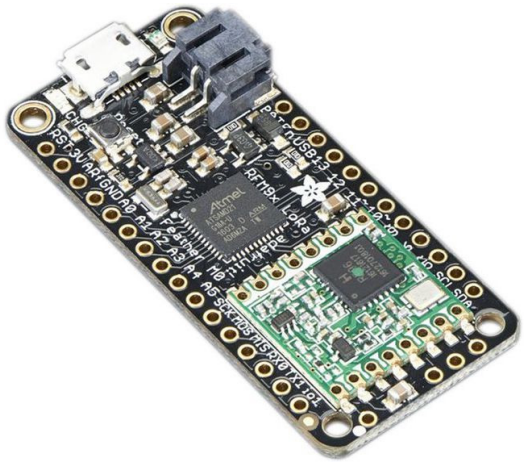
LoRa MAC features no security

Plaintext packets, we just need to know the channel

Channel hop, display packet

Capture to disk

Implemented in the Feather M0 and RFM95



LoRa Attack Example 1

LoRa/LoRaWAN networks are often comprised of small, distributed devices that have the keys resident in RAM. If we could use the hardware hacking techniques that we've learned to recover keys from the device, all we'd need to do is verify the presence of one and locate it; and use the recovered keys to decrypt our LoRa packets.

Of course, if we want to decrypt packets for a network, we need to obtain them through some sniffing first. Specifically, Identifying the PHY layer LoRa packets is relatively easy. The are in plain text, and easily decipherable, se just need to hop through all of the available channels looking for the presence of packets. Once found, display the packet to some observable event, and record it to disk for further analysis.

We can create just such a sniffer using the Adafruit Feather M0 platform featuring the RFM95 (900MHz) LoRa radio, or the RFM98 radio with some slight code modifications.

LoRa Sniffing

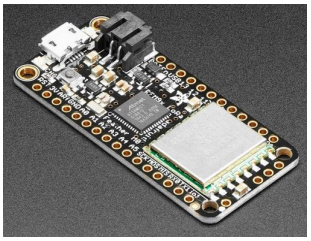
Hardware based LoRa MAC sniffer

Channel hops and records packets to SDcard

Discovery displayed to serial console

Based on Feather TFT Sniffer by Improbable Studios

- No TFT needed



```
LoRa init OK        Freq: 8680 (0000)   Freq: 8680 (0000)
MS,Freq,FreqPacketCount,RSSI,From,To,MsgId,Flags,Ack,DataLen,DataBytes,Data
Freq: 9150 (0000)   Freq: 9150 (0000)
Freq: 9039 (0000)   Freq: 9039 (0000)
Freq: 9041 (0000)   Freq: 9041 (0000)
Freq: 9043 (0000)   Freq: 9043 (0000)
Freq: 9045 (0000)   Freq: 9045 (0000)
```

https://github.com/haxorthematrix/Feather_TFT_LoRa_Sniffer

LoRa Sniffing

Using the Adafruit Feather M0, we can create as hardware-based sniffer based off of the Feather_TFT_LoRa_Sniffer code (https://github.com/ImprobableStudios/Feather_TFT_LoRa_Sniffer) From Improbable Studios/ The initial implementation can channel hop, find packets and write them to an onboard SDcard. This does require us to have the addition of a somewhat obscure TFT screen with integrated SDcard so an updated version by @haxorthematrix logs the discovered packets to the built-in serial console. We can access the logged data using any serial console with the updated code but lose the ability to store it to SDcard.

LoRa Attack Example 2

LoRa networks have lots of small, distributed devices.

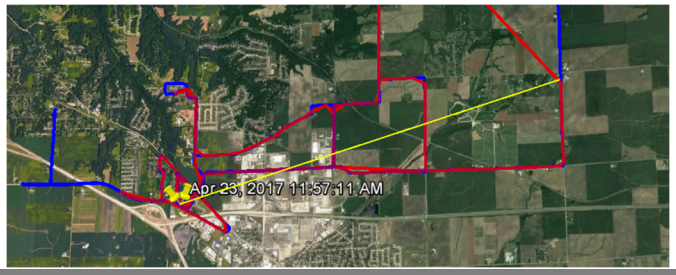
Encryption key is resident in RAM

Find a device you'd like to "audit".

- Given LoRa range, this is the hard part!

Leverage device to retrieve encryption key from hardware.

Use recovered key to access network or decrypt all previously captured traffic.



LoRa Attack Example 2

We can confirm that devices are present, we now just need to find one! Those small distributed devices store the keys on board, and once we've been able to analyze the firmware, we should be able to use those keys to decrypt packets, or even participate in the network. We can surmise that the network key is stored in flash until the device boots, where it is moved to RAM and then used to encrypt and decrypt traffic on the network. If an attacker is able to capture the contents of flash or RAM, they can use this information to search for the encryption key by repeatedly attempting to decrypt a valid, encrypted packet.

Given that LoRa networks can be quite expansive in distance, this is both a challenge and a curse! The range can make it difficult to accurately recover a device, but once we do and can participate locating the attacker who has recovered keys and is injecting data is equally as hard.

LoRa Packet Capture

Need to capture packets!

gr-lora with an appropriate SDR

- TX with Pycom LoPy, Adafruit Feather 32u4
- RX with HackRF

Capture with Wireshark with gr-lora UDP output

```
$ lora_receive_realtime.py
40F17DBE490002000 1954378762B11FF0D
$ lora-packet-decode --appkey ec925802ae430ca77fd3dd73cb2cc588 --nwkkey
44024241ed4ce9a68c6a8bc055233fd3 --hex 40F17DBE490002000 1954378762B11FF0D
```

https://github.com/rpp0/gr-lora

LoRa Packet Capture

We can use gr-lora with an SDR and some other hardware to fully interact with and decrypt LoRa packets.

gr-lora can receive with a HackRf, and we can transmit with some programming to a Pycom LoPy, or Adafruit Feather 32u4 device. Gr-lora can also send received packets directly to Wireshark over UDP; all we need to do us capture the UDP packets with Wireshark on an available network interface.

Additionally, using the lora_receive_realtime.py python script, we can capture packets and pass them to lora-packet-decode with the appropriate, recovered keys, to manually decrypt packets for us.

Summary

LoRa is a growing low-power wireless protocol for the IoT market.

LoRa has a complex stack that can be secured approximately.

Sniffing and Key recovery possible with firmware analysis, dedicated hardware

LoRa distance makes device locating challenging.

Growing adoption will see the development of new tools.

Summary

LoRa as an IoT wireless protocol is gaining traction for long distance communications in IoT deployments. With a complex stack, it is easy to make mistakes in implementation, however this complexity can add to its security when configured properly, though appropriate encryption, keying, and some obscurity with channel width and selection.

Even when configured properly and secured with encryption, key recovery is possible though hardware and firmware analysis of an end device. Specialized hardware can be used to sniff traffic from the air.

Due to the extended distance in which LoRa can communicate, this makes locating rogues, nodes, or attackers challenging. While LoRa is still new from an adoption perspective, more tools will become available over time as attackers take an increased interest in the protocol in order to gain access to data or networks.

Course Roadmap

Introduction to IoT Network Traffic and Web Services

Exploiting IoT Hardware Interfaces and Analyzing Firmware

Exploiting Wireless IoT: Wi-Fi, BLE, Zigbee, LoRa, and SDR

SECTION 3

1. **Wi-Fi**
   Exercise: Wi-Fi PSK Cracking
2. **Bluetooth Low Energy**
   Exercise: BLE Device Interaction
3. **Zigbee**
   Exercise: Zigbee Traffic Capture
4. **LoRa**
5. **SDR**
   Exercise: Conducting a Replay Attack on IoT

This page intentionally left blank.

What Is Software Defined Radio?

SDR:

- Is not limited by single-purpose hardware
- Has no preconceived notion of purpose

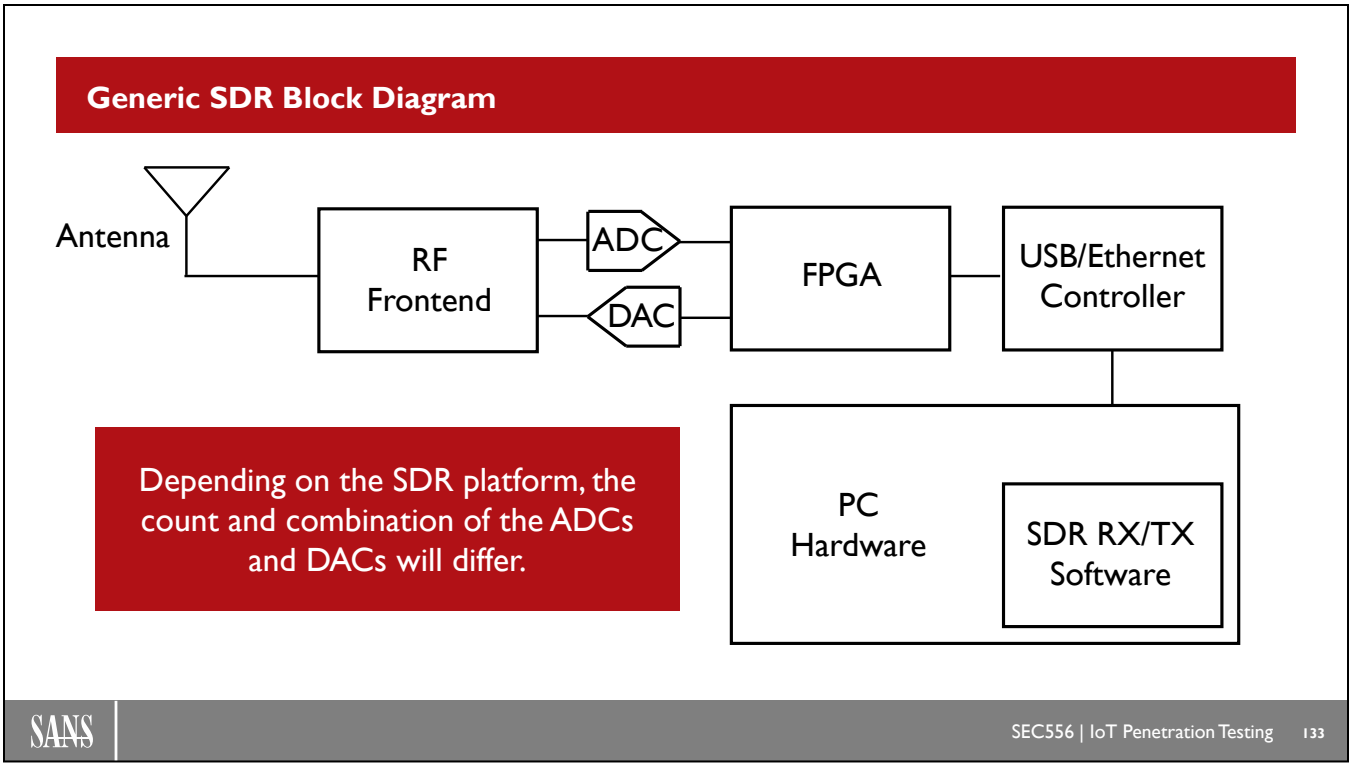
Requires programming and real-time processing

- Various software packages with direct access
- Multipurpose tool such as GNU Radio

SDR is hugely flexible but requires lots of work

What Is Software Defined Radio?

Software Defined Radio (SDR) is a platform where instead of developing modulation mechanisms in hardware, developers can write software to process and interpret signals received from the RF spectrum. Traditional wireless devices, such as Wi-Fi cards, Bluetooth adapters, GSM receivers, and other proprietary systems are designed to be cost effective and limit accessibility into the wireless spectrum while providing the functionality needed for the product design. By contrast, SDR technology does not make any assumptions as to how the user wishes to interpret the spectrum for receive or transmit paths. By default, SDR technology does not do anything to process signals, allowing users complete freedom in developing software to interpret signals as they see fit. In the commodity SDR hardware available for purchase today, the end user has access to a wide range of frequencies, from 1 MHz all the way to 5.9 GHz, depending on the SDR selected.



Generic SDR Block Diagram

Generically, the SDR hardware interacts with a host system over a USB 2.0, 3.0, or Ethernet connection, depending on SDR selection. Connecting to a Field Programmable Gate Array (FPGA) that includes software for encoding signal information into binary format, the FPGA interacts with a number of analog to digital converters (ADC) and a number digital to analog converters (DAC). The ADCs and DACs connect to a variety of receive and transmit frontends.

For the process of demodulating traffic, a piece of host software running on PC hardware will tune the receive board to a given frequency and gain level. When a signal is received, it is passed to the ADCs that feed the data to the FPGA. The default FPGA software that is supplied with the SDR converts the signal to a byte of data with the least-significant bit set or cleared.

Why Software Defined Radio?

Universally accessible, new access to wireless spectrum

Freedom of software for access to RF spectrum

- Not limited by single-purpose hardware

Wide frequency range

Exposes protocols previously inaccessible to attackers

Rise of IoT

The rise of the Internet of Things (IoT) has seen the influx of proprietary wireless technologies. In order to make them easy to use and keep costs down, security implementation is often weak or nonexistent. Evaluating these systems can reveal significant issues.

Why Software Defined Radio?

Software Defined Radio (SDR) is a platform where instead of developing modulation mechanisms in hardware, developers can write software to process and interpret signals received from the RF spectrum. Traditional wireless devices, such as Wi-Fi cards, Bluetooth adapters, GSM receivers, and other proprietary systems are designed to be cost effective and limit accessibility into the wireless spectrum while providing the functionality needed for the product design. By contrast, SDR technology does not make any assumptions as to how the user wishes to interpret the spectrum for receive or transmit paths. By default, SDR technology does not do anything to process signals, allowing users complete freedom in developing software to interpret signals as they see fit. In commodity SDR hardware available for purchase today, the end user has access to a wide range of frequencies, from 1 MHz all the way to 5.9 GHz, depending on the SDR selected.

SDR technology has the potential to be advantageous for innovative technologies—including cognitive radio, where a wireless device is able to dynamically adapt to the current RF spectrum and protocols in use. SDR technology also represents new opportunities for an attacker to access the previously inaccessible wireless spectrum.

What Is the Threat?

Proliferation of TX/RX devices

VaporTrail, from Larry Pesce and Faith Alderson of InGuardians

- Low bandwidth exfiltration of data over FM
- Using a Raspberry Pi as a transmitter
- Reception via low-cost receive-only RTL-SDR

How can IoT devices be discovered in the environment?

How can IoT devices become *unintentional transmitters*?

How can interaction with ”non-standard” RF cause mayhem?

What Is the Threat?

Additionally, it is worth noting that radio may help aid in the ‘typical’ objectives of a penetration test or red team operation.

One example of this is a tool called VaporTrail, released by Larry Pesce and Faith Alderson of InGuardians in 2017. VaporTrail allows for easy exfiltration of data over FM frequencies using a Raspberry Pi and RPITX. While this will require a receiver at the other end to decode the data, this can be accomplished for low cost using a receive-only SDR unit like an RTL-SDR.

This solution can only transmit at 2000 bits per second, so this will not meet a need for high throughput. However, it would be perfectly serviceable for transmitting small amounts of data, such as password hashes obtained via tools like Responder.

Pesce and Alderson developed the tool as a method to perform exfiltration of data during a penetration test while going largely undetected, due to a distinct lack of monitoring and detection of changes in RF baselines. While developed with good intentions, the toolset is now available to malicious actors as well.

SDR can help enhance detection capabilities where traditional network monitoring techniques such as DLP and IDS fail.

VaporTrail is available at https://github.com/inguardians/VaporTrail.

Choosing an SDR

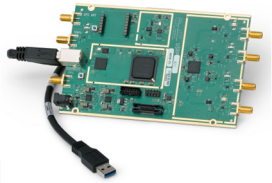
Proliferation of SDR platforms

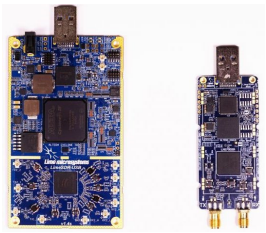
Use the right tool for the right job.

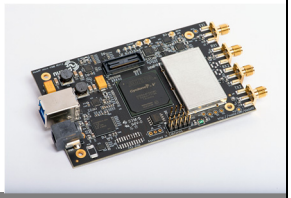
Many decisions can affect selection.

- TX capable, or RX only
- Bandwidth for RX/TX
- Cost
- Tool capability and ease of use

TX capable SDRs have great power (and great responsibility).







Choosing an SDR

With the proliferation capable, affordable SDR platforms available, it is understandable that there is confusion when selecting an appropriate platform. We can very easily get carried away in our selection and cost for any given project, but they may be overkill.

Overall, we should consider using the right tool for the right job, which may lead us to obtaining more than one SDR platform over time. We should carefully consider if we need to transmit (or receive only), the capabilities of the receive and transmit bandwidth within the SDR and our targets, the overall cost, and how well the selected platform toolchains integrate with the tools we intend to use for capture and transmit.

Be aware that many of the interactions we will have with IoT devices are in the Industrial Scientific and Medical frequency bands, meaning they are available for use by consumers without the need for obtaining specific licensing for transmission. With SDR, we are not limited to where we can transmit, and we can use them to transmit outside of the ”free” ISM bands. This capability comes with great power and responsibility, as transmitting in areas of the spectrum in which we are restricted, or require specific licensing can get us in trouble, and in violation of various laws.

We’ll be offering some reasonable examples is SDR models for working with IoT.

HackRF One

Two varieties, HackRF One readily available

RX and TX, $300

Half-duplex 1 MHz–6 GHz, 20 MHz width*

Open source, several derivatives and add-ons

- CCC rad1o
- Portapack

Huge community support

HackRF One

Currently, two models are available. All models feature the ability to do half-duplex TX and RX. The HackRF was created and developed by Michael Ossman in full support of the open-source community.

The Hack RF Jawbreaker uses USB, has a 100 MHz–6 GHz frequency range, is priced from Mr. Ossman's Kickstarter campaign, and has an extremely limited used market. The Jawbreaker was also able to benefit from support from DARPA's Cyber Fast Track Program, which allowed Mr. Ossman the resources to develop the device.

The Hack RF One provides some updates to the Jawbreaker concept and was released via Kickstarter at $199 for early adopters. It was slated for release in early 2014. Early beta models were available for demonstration at Shmoocon 10 in early 2014 and are now readily available from many online retailers and select brick-and-mortar locations. The HackRF One can safely tune between 1 MHz and 6 GHz.

Mike Ossman has had success with Kickstarter campaigns and designing radio devices, including the Bluetooth Ubertooth One and YARD Stick One. Mr. Ossman's device following is due in part to his commitment to hackability and the community that uses his designs. There is huge community support and either direct support for HackRF or support for all sorts of applications via shim/additional driver support (often developed by the community). The HackRF One schematics are freely available and have been used to produce derivative works, such as the Rad1o project at the 2015 Chaos Communication Congress.

RTL-SDR DVB-T Tuners

Eight+ varieties, most are readily available

RX only, $20–$35

Half-duplex, 24 MHz–1.7 GHz common, 2.4 MHz bandwidth

Closed source hardware

- Some custom designs with many improvements
- Heatsinks, temperature stabilized oscillators

Custom drivers for Windows, Linux, macOS

RTL-SDR DVB-T Tuners

These tuners were originally intended to be digitally broadcast OTA TV receivers in a small package for your PC. The frequencies and channels supported are only EU based, so they are of limited utility value in the US. However, by leveraging some hardware hacking techniques, it was discovered that the hardware was actually a limited-capability SDR and radio tuner chip generally supporting frequencies between 52 and 2200 MHz. A review of the chip documentation revealed capabilities of the devices and that they were, in fact, low-power SDRs with all of the DSP work not done in an FPGA but by the resources of the host PC. The manufacturer-provided tuner software handled the demodulation of DVB broadcasts.

Drivers were reverse engineered and newly written to provide direct access to the hardware, eliminating the need for the (mostly useless) manufacturer-provided software.

Many of the tuners will receive well outside of their manufacturer-documented frequency ranges but will vary per device and the specific manufacturing tolerances. Unfortunately, these devices are RX only. This limitation is counteracted by the extreme versatility of these devices (especially at the price point of between $20 and $40).

Most feature some sort of external antenna connector (MMCX or PAL), and in most cases, the provided antenna is mostly useless. Inexpensive antennas can be obtained through a variety of online retailers and can be tuned for specific purposes.

Antenna Length

For transmit (TX) and receive (RX) antenna length is important

RX is best when the antenna matched to the wavelength

- RX works with any random length
- Most efficient when tuned properly

For TX, antenna length is extremely important!

- Too long is highly inefficient
- Too short results in VSWR problems

If VSWR ratio is too high damage to the transmitter can occur

Full Wave Total Length in Feet = 984 / Frequency in MHz
Full Wave Total Length in Meters = 300 / Frequency in MHz

Antenna Length

In order to have the most efficient reception and transmission, it is important to have the antenna tuned for the desired frequency range. This length is determined by the frequency, and therefore the RF wavelength based on a specific formula: The total length of the antenna in inches is equal to 984 divided by the frequency in MHz. For metric conversion, the formula for antenna length in centimeters is 300 divided by the frequency in MHz. Antenna calculation *can* be incredibly complex depending on the design and environment, but this formula is suitable for most applications (https://ham.stackexchange.com/questions/283/calculating-antenna-length-on-the-fcc-exam-vs-in-reality).

For reception (RX), any random length antenna will work, but it will not be as efficient as a properly tuned antenna.

For transmit (TX), antenna length is extremely important. An antenna used for TX that is too long is highly inefficient, resulting in significantly reduced range for reception. However, if an antenna is too short during transmit, the RF energy trying to "escape" from the antenna cannot efficiently, and the antenna captures remaining energy from the wave, folding it back towards the transmitter instead of in the air.

If, during transmit, an antenna is too short and the resulting VSWR ratio is too high RF energy is reflected back to the transmitter. This reflection, depending on power and duration can permanently damage the transmitter.

In many cases, an antenna at the appropriate length is too large to be practical, so using an evenly divisible portion of the antenna length is appropriate: often ½ and ¼ wave antennas are used. Each element, or portion of the dipole antenna, divides the antenna length in half.

Basic Visualization

To discover signals, the best way is to see and hear the signals.

Visualization of the spectrum

Some basic options include:

- gr-phosphor
- GQRX

Both options support Linux, macOS, and Windows.

We'll be examining the features and capabilities of the spectrum visualization tools on the next few slides and interact with some during lab exercises.

Basic Visualization

Because wireless signals, represented as radio frequency (RF) energy, are colorless, tasteless, and odorless, it can be very difficult for us as humans to detect their presence. However, by using the capabilities of an SDR device, we can use the full bandwidth of an SDR receive function to convert the presence and strength of RF energy to a visual representation, making it much easier to observe and locate in the spectrum.

In this section, we will examine gr-phosphor and GQRX to visualize the RF spectrum for signal hunting, both of which have varying levels of support for Linux, macOS and Windows.

gr-fosphor

FFT and historical waterfall display

Decidedly Hi-Fi

- High resolution display based on OpenCL
- Quick view, time-based observation

Control via GUI slider, limited options

No demodulation, plugins or audio output

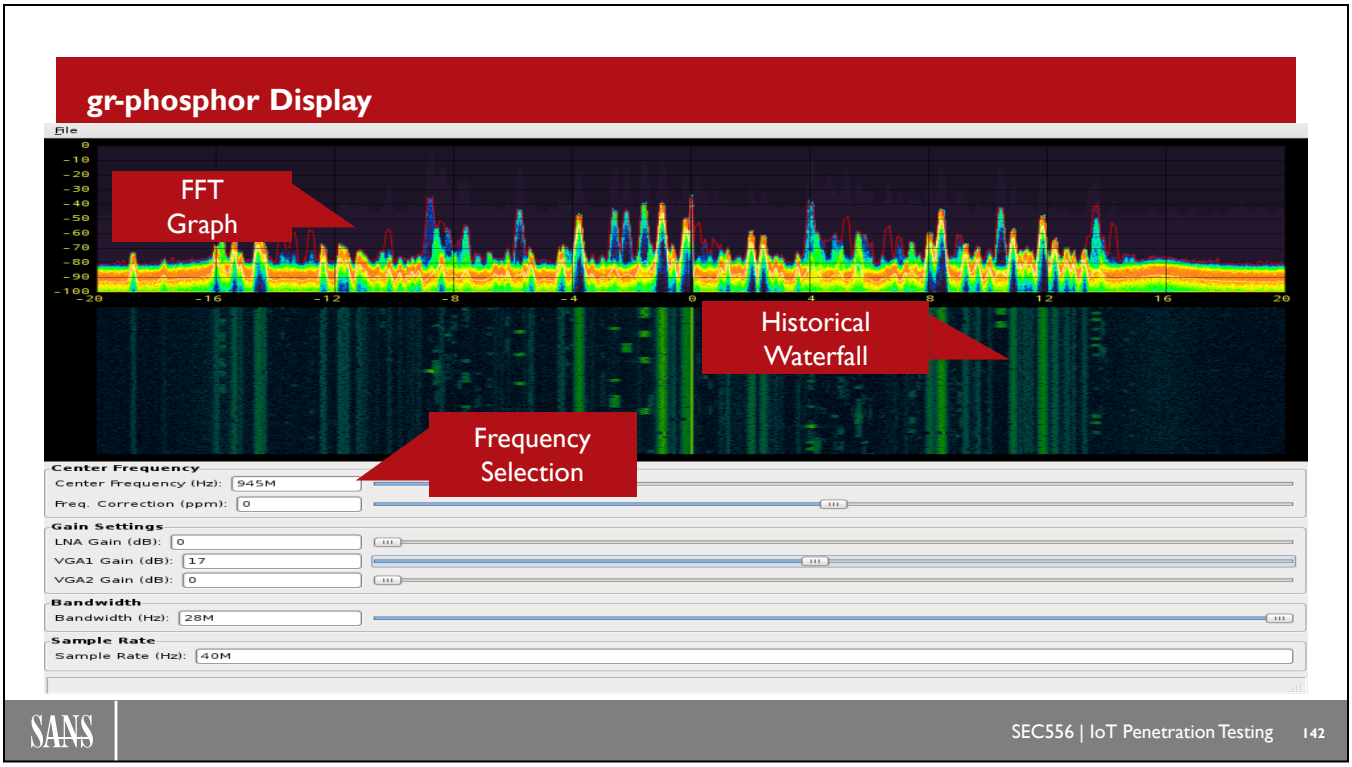
Operates standalone or as a GNURadio block

gr-fosphor

As a polar opposite to tools such as the lo-fidelity retrogram~rtlsdr, gr-phosphor takes a high-fidelity approach to visualization of an FFT and historical waterfall display. The high-resolution display is accomplished through OpenCL rendering, making discovery and detailed visualization of signals quite easy in real time (FFT), or over time (historical waterfall).

Gr-phosphor relies on a GUI slider (and text input fields) for control, yet also does not feature any other expanded capabilities, such as audio output, demodulation options, or plugin architecture.

While gr-phosphor can operate as a standalone discovery module, it can also be integrated into any other GNURadio projects where signals visualization is required.



gr-phosphor Display

The gr-phosphor features an FFT graph, and historical waterfall display in high resolution on the top portion of the output. On the bottom portion of the display, the text entry fields and sliders can be used to reconfigure the display in areas in which to focus investigation and discovery.

GQRX

Multiplatform, open-source SDR application
- Linux: apt-get install gqrx
- macOS (buggy): ports install gqrx
- Windows via a PothosSDR bundle

Real-time FFT display

Historic waterfall display

Plugin support
- Built-in for basic operation, demodulation
- Community-driven add-ons for advanced features

Recording of samples to disk

GQRX

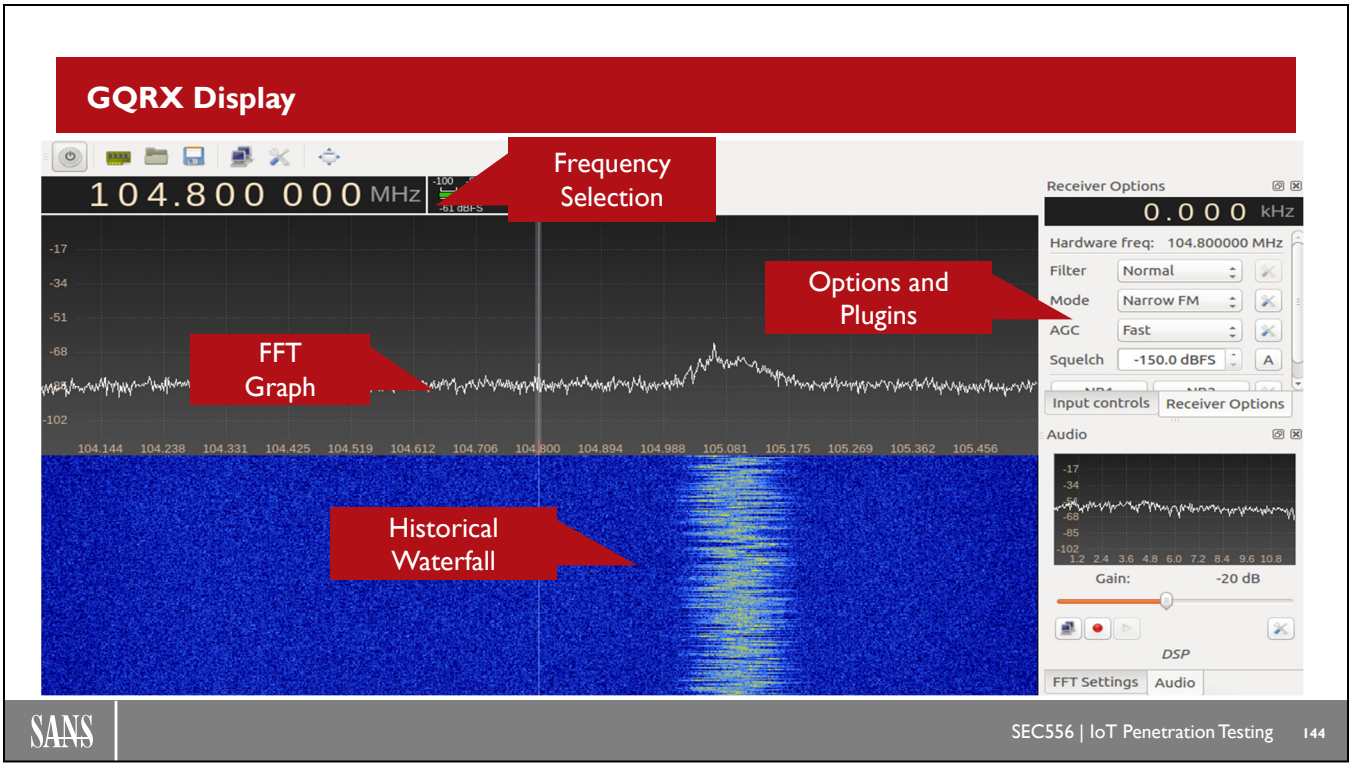
GQRX is a simple-to-use application to interact with, tune, and provide basic demodulation to RF signals. While GQRX is primarily a Linux application, it can be installed on the Mac through Portage (or via a binary distribution). However, this author has found that the macOS version is very buggy and erratic in operation.

We can use the interface to manually tune to specific frequencies and scroll through the Fast Fourier Transform (FFT) display to change the frequency and pick individual signals to focus on. Additionally, GQRX features a waterfall display. Where the FFT display is in real time (providing a point-in-time display of the signals), the waterfall display is a historical record of the changes in the FFT display. This gives us the ability to observe the changes in signals over time.

It can perform some basic demodulation (AM, NFM, WFM, CW, SSB, etc.), but the magic happens with the plugin support so that developers can add their own features, including additional demodulation and decoding, scanning utilities, and even frequency memories. GQRX also features the ability to record audio samples of the tuned signal to disk as well as stream to the network over UDP.

GQRX can be obtained from the following locations:

- For Linux and macOS: http://gqrx.dk/download
- For Windows: https://github.com/pothosware/PothosSDR/wiki/GQRX



GQRX Display

The GQRX interface is comprised of two major parts: On top, a Fast Fourier Transform (FFT) display showing real-time peaks in transmission strength and, on the bottom, a waterfall display (a flat rendering of signal strength) indicating signal strength over time.

In addition to these two parts, we can tune using the frequency selection at the top to refocus our display below. On the right, we can demodulate, manipulate, record, and use various plugins for our selected signal.

Capture and Replay Attacks

Using an SDR device with TX capabilities, it is possible to capture and then replay signals.

- Unmodified, no change in data
- Potential for trimming for brevity with Audacity

Works exceptionally well against protocols that:

- Do not use rolling codes.
- Do not perform mutual authentication or per-packet authentication.
- Have only one-way transmitters—i.e., one device is TX only, the other is RX only.

Replay can't be that bad, can it?

Capture and Replay Attacks

If we add the ability to transmit with an SDR device into the mix of analyzing signals and systems, things can get interesting very quickly, even without the need to dive deep into demodulation, counting bits, or reverse engineering protocols. In some cases, we can capture signals with our SDR device and replay the raw data back into the air on the same frequency and produce interesting, potentially undesirable results. In the case of a replay, it is a true replay situation: Capture, optionally trim to isolate signals, and replay unmodified.

Capture and replay attacks work against several scenarios when the impact is not fully considered—in some cases, before the advent of high-fidelity technology that could perform these attacks. These attacks do still exist in modern products, often when one of several conditions arise: The transmissions are not unique; the transmissions are not using rolling codes or encryption; the devices do not perform mutual authentication or per-packet authentication; or the interacting radios are performing only one-way communication, making the correction of the other two conditions impossible.

Of course, the manufacturers of the devices that are subject to these issues took this into account at lower layers of the protocol that we have yet to decode. Didn't they? What could possibly go wrong?

rtl_433

Generic 433 MHz ISM band receiver for the RTL-SDR

Highly capable signal analysis

Fingerprints of known transmissions

- 100+ known devices
- Automatic, live demodulation and decode

Record and playback raw samples

- Great opportunity to replay with TX capable SDR

Capable analysis for message recovery, custom decoders

rtl_433

Rtl_433 is a generic receiver application for use with the RTL-SDR that specifically tunes to 433MHz in the ISM band, with the intent of discovering RF energy. If RF energy is detected it will attempt to demodulate and decode the signals. Upon demodulation, and decoding, the binary data is compared to more than 100 previously decoded data types and formatted with human-readable output, identifying the device that sent the data. Not all of the device fingerprints are enabled by default, as they are subject to false positives.

While rtl_433 will perform live demodulation and decoding of data, it can also write the captured data to disk. These captures can be for the entire session in which rtl_433 was run, or automatically split by portions of data received during the session. In either case, rtl_433 is able to replay those samples, not through the air, but through the internal analysis engine, with an attempt to decode at a later date.

Once captured, it is possible to replay the samples with a transmit capable SDR, such as the HackRF.

Rtl_433 can be found at https://github.com/merbanan/rtl_433.

rtl_433 Example (1)

```
$ rtl_433
<...trimmed for brevity...>
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
time      : 2021-04-28 13:04:32
model     : DSC-Security id        : 2291244
closed    : 1            event     : 0            tamper    : 0            battery_ok: 1
xactivity : 0            xtamper1  : 0            xtamper2  : 0            exception : 0
esn       : 22f62c       status    : 195
status_hex: c3           Integrity : CRC
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
time      : 2021-04-28 13:14:36
model     : DSC-Security id        : 2696754
closed    : 1            event     : 0            tamper    : 0            battery_ok: 1
xactivity : 0            xtamper1  : 0            xtamper2  : 0            exception : 0
esn       : 292632       status    : 195
status_hex: c3           Integrity : CRC
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
^CSignal caught, exiting
```


rtl_433 Example (1)

In this demonstration of rtl_433 using no additional options, it tunes the RTL-SDR to 433.92 MHz, captures signals, and demodulates and decodes automatically based on all known, reliable decoding types.

In this case two different separate devices were found: Two different DSC Contact sensors, with individual IDs of 2291244 and 2696754. These DSC Contact sensors have been linked to ADT burglar alarm system door and window open/close sensors. Depending on how the alarm system interprets results, this may present an opportunity for capture and replay, resulting un undesired results to the system. We'll see another occurrence of these sensors later on in this section.

rtl_433 Example (2)

```
$ rtl_433 -a4
<output trimmed for brevity>
Iteration 1. t: 5    min: 2 (1)    max: 9 (1)    delta 0
Pulse coding: Short pulse length 2 - Long pulse length 9
Short distance: 2, long distance: 0, packet distance: 2
p_limit: 5
bitbuffer:: Number of rows: 2
[00] {1} 00 : 0
[01] {1} 80 : 1
signal_bszie = 131072  -      sg_index = 2883584
start_pos    = 2613856  -   buffer_size = 3145728
*** Saving signal to file g014_433.92M_250k.cu8
*** Writing data from 2613856, len 131072
^C
$ rtl_433 -X 'n=EV1527,m=OOK_PWM,s=160,l=472,r=4796,g=484,t=124,y=0'
<output trimmed for brevity>
2021-04-26 16:35:06 :     EV1527   :       1       :       1       :
        [       :       25       :       fac9b38]        :
        [{25}fac9b38]
```

HACKRF?

rtl_433 was designed for the RTL-SDR, but can work with the HackRF and others with the Soapy framework complied in. The addition of a –d "" to use the Soapy framework is required, as well as some SDR specific options for gain, etc.

rtl_433 Example (2)

In this additional example, using the –a and –t options will attempt to locate and analyze unknown transmissions, or ones that do not match currently defined fingerprints. In this case, *rtl_433* has discovered and captured an unknown signal. By performing additional analysis with *rtl_433* using the recorded file (in this case `g014_433.92M_250k.cu8 captured in the first step`) it can offer some suggestions for creating custom decoders with the –X option.

In our custom decoding example, using suggestions made by *rtl_433* we are accurately able to decode some data, sent by an EV1527 RF chipset. This is advantageous, in that *rtl_433* features an EV1527 decoder, but it is unable to decode the data from our hardware, as the manufacturer has apparently implemented it in s "non-standard' manner.

Rtl_433 is specifically designed for use with the RTL-SDR line of SDRs, but with some additions at time of compile and the addition of the Soapy framework prerequisites, we can use *rtl_433* with a hackRF!

HackRF Command Line

Simple tools for capturing and replay with *hackrf_transfer*

Transfer of IQ data from radio to file (and back)

Set sample rate (-s), frequency (-f), file output (–r)

Transmit adds file input (-t), enable amplification (-a), set gain (-x)

```
$ hackrf_transfer -s 2000000 -f 433920000 -r EV1527.8s
call hackrf_set_sample_rate(2000000 Hz/2.000 MHz)
call hackrf_set_freq(433920000 Hz/433.920 MHz)
Stop with Ctrl-C
 3.9 MiB / 1.001 sec =  3.9 MiB/second
^CCaught signal 2
Exiting...
$ hackrf_transfer -s 2000000 -f 433920000 -t EV1527.8s -a 1 -x 24
call hackrf_set_sample_rate(2000000 Hz/2.000 MHz)
call hackrf_set_freq(433920000 Hz/433.920 MHz)
Stop with Ctrl-C
 3.9 MiB / 1.003 sec =  3.9 MiB/second
^CCaught signal 2
Exiting..
```

Capture

Replay

HackRF Command Line

If we have performed capture and analysis with *rtl-433* it would be helpful to replay the captured data in order to see if capture and replay based attacks are possible. While capture and analysis with *rtl_433* is helpful)even capturing with the HackRF), we're largely left with no way to replay with the tool.

If we have identified an appropriate target for capture and replay, we can perform the capture and retransmission of the data directly with the HackRF, using the utility *hackrf_transfer*. While not intended by the tool's author to be used for such purposes of capture and repay attacks, it is still effective, nonetheless.

In this capture example with *hackrf_transfer* we are defining a few options: -s sets our sample rate, -f sets our frequency, in this case to the common center frequency for 433 MHz ISM band devices, and –r for our output file on disk. We've chosen to use the *.8s* file extension, to indicate that the captured data is in the 8-bit signed IQ format, the default for *hackrf_transfer*. On the replay side, -s and –f are the same (to match our capture command) but we replace the –r with a –t to read from a file, as opposed to write. We also append –a 1 to turn on the internal HackRF amplifier to increase the signal strength and distance, and –x to set some appropriate level of output power with gain.

GNU Radio

Open-source radio construction in software

- Comprised of Python and C code blocks
- GNU Radio Companion (GRC) for drag-and-drop construction

Complex, many options

- Predefined blocks for input, output, filtering, modulation, demodulation, complex math, and more
- Support for most SDR platforms

If code blocks do not exist, we can create them!

GNU Radio

GNU Radio is an open-source radio construction application originally created by Ettus Research, and now largely supported by the community. GNU Radio leveraged a mix of Python and C to create hundreds of functions for radio creation. Creating radios purely with code is quite complex, especially with more complex, in-depth features; GNU Radio Companion (GRC) provides easier access to the code blocks by providing a drag-and-drop interface that is relatively easy to use. There is still some complexity in determining the blocks that are needed to construct a radio, but the maintenance of the underlying code interaction is removed from the user by the graphical interface.

GNU Radio is quite complex in that there are many code blocks that can be stitched together, requiring an in-depth knowledge of RF processing and background to be effective, especially when complex RF processing is needed. These multiple code blocks include the ability to transmit and receive from most SDR hardware, perform file read and write operations, filter, and apply complex mathematical operations to the various signals. Additionally, Ettus and the community have developed many different modulation and demodulation blocks for ease of use (FM, AM, GFSK, QPSK, etc.) to make construction easier.

In some cases, we may find that the particular code blocks do not exist for the signal analysis or creation that we are trying to accomplish, but due to the modular and open nature, we can create our own compatible code blocks.

Capture and Replay with GRC

GRC is complex, but simple capture and replay is not.

- Ins and outs of GRC

Source is an input

- HackRF, RTL-SDR, generic SDR, file

Sink is an output

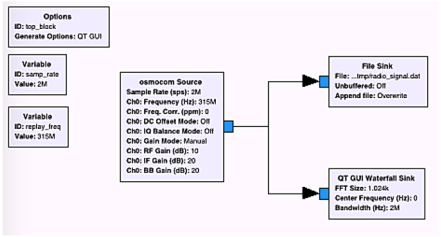
- Also, HackRF, RTL-SDR, generic SDR, file

Add some basic visualization for RX and TX confirmation.

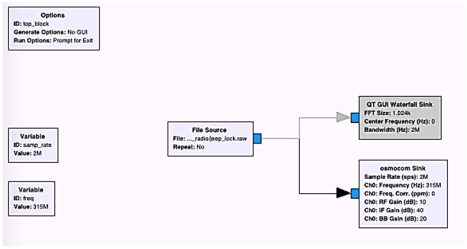
- Simple FFT, gr-fosphor

Variables defined in blocks for simplicity

Capture



Replay



Capture and Replay with GRC

While *hackrf_transfer* is an effective tool to use from the command line, sometimes we may want to consider performing additional actions to the signals or doing so in a more visual manner. To accomplish those goals, we can use GNURadio and the GNURadio Companion (GRC) GUI. While GNURadio can be very complex to use, basic replay-based attacks are quite simple to implement with just a few components.

In the simpler form with GRC we need to define some inputs and outputs and connect the dots between the two. The "source" in GRC is the input, and the "sink" is the output. In our capture example the source is the HackRF via the osmocom Source, and the output is a File Sink for writing to disk. On the replay, the roles are reversed: the File Source is reading from disk, while the osmocom Sink is our output for the HackRF.

In our example here we have used some additional components to make our flowgraph easier to use and perform validation. We have defined a few variable boxes on the left for our frequency and sample rate, and then reference those variable names within the Sources and Sinks as needed. We've also added some visualization on the output or Sink side, so that we can verify via an FFT graph that signals are being captured and sent.

From RF to Bytes

More in-depth RF analysis required

- Modulation type: AM, FM, OFDM, PSK, OOK, ...
- Encoding type: Manchester, BPSK, BOC, ...
- Recovering binary data

After analysis, can we create our own transmissions?

- Can we modify previously captured transmissions?

Figuring those parts out can be complex, time consuming

The entire RF to bytes process is not for the "faint of heart" and should used when other options have failed, or transmissions need to be modified.

From RF to Bytes

We have covered several projects that use Software Defined Radio, relying on the work of others to perform the demodulation, decoding, and analysis of the signals. Those project authors have likely spent many hours performing complex analysis of the received signals. When we are presented with an unknown signal, this now becomes a task that we need to undertake.

This task of taking an RF transmission and turning it back into something that humans or a computer can understand and interact with can be an arduous process. In order to start, we need to answer several questions and perform several tasks to recover data:

- Determine the modulation type, so that we can determine how the wave form was used to transfer data.
- Determine the encoding in use in order to determine how the spacing of the wave form was used to indicate data.
- Convert demodulated and decoded data to binary data.
- Determine the meaning, format, and protocol that the binary data represents.

Ultimately the intent would be to understand the output of the received data, so that we can interpret the protocol data in order to determine "which bits to flip" for our own transmissions, with the intent of performing some action to the transmitting device.

This discovery and analysis process can be time consuming when little to no data about the transmission is available. This means we may need to perform repeat analysis, using trial and error to recover correct data at each step of the process.

FCC ID Lookup

Need to determine frequency, other information

TX in the US requires government testing, approval

- US labeling, worldwide applicability

Device labels include an FCC ID

- Indicator to FCC records for spurious emissions testing
- Documents often reveal implementation details
- Some data may be obscured to protect intellectual property

Query for FCC IDs at https://www.fcc.gov/oet/ea/fccid

FCC ID Lookup

In order to simplify the process of discovery, obtaining documentation on the device can reveal information to shorten our analysis process. This documentation can include frequency ranges, modulation types, encoding, and even a description of the protocol. We may not always get this lucky however, as some information on our target device may be obscured from our discovery methods as it may be considered Intellectual Property.

If our device is manufactured for use in the US and emits RF energy, it must be tested for compliance and documentation provided to the FCC for licensing. This licensing information must be included on the device itself; while the license is US based, often the device is marketed worldwide with the same features and technology without any change in the packaging. Similarly, Industry Canada (IC) requirements are similar for testing and labeling and can also be a font of knowledge.

Labeling on the target device should indicate a unique identifier that can be looked up at the FCC's website, revealing the supporting data for the FCC, so that the manufacturer can offer the device for sale. This often times reveals data critical to our signal's recovery efforts.

Note that, while the FCC's website for lookup is not particularly memorable and difficult to find on the FCC website, fcc.io can proxy the request on our behalf. This comes with added challenges—we are effectively providing notification in our interest in a specific device to Dominic Spill (@dominincgs) the creator of fcc.io.

URH

Universal Radio Hacker

Observation of waveform transition, time

Recovery of symbols per second (data rate)

Application of several common demodulation types

Recovery of binary data, convert to text

Reads from common capture data formats or direct from SDR

Transmit with HackRF, others

- Possible to modify analyzed data
- Great for unmodified replay attacks too!

URH

Universal Radio Hacker (URH) can load samples and display the waveform transition over time, but with significantly more features. At the outset, URH can obtain samples directly from an appropriate SDR.

While we can also recover the symbols per second on a transmission with URH, we can also apply several basic methods of demodulation and decoding, in order to recover the binary for a transmission, which can be converted to text where appropriate. Additionally, the conversion to binary can be modified and retransmitted with an appropriate SDR or Rfcat dongle.

Overall, URH features far eclipse that of similar tools, such as inspectrum with respect to data recovery, modification, and re-transmission.

URH Process (1)

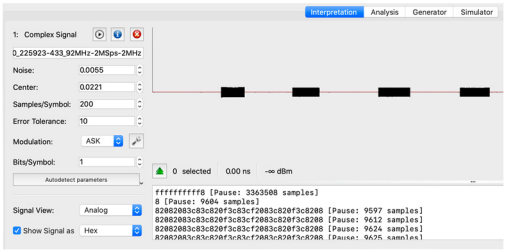
Record, Save, Analyze

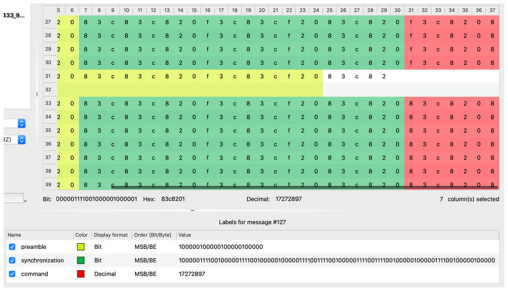
Signal autodetection quite reliable, but not foolproof

- Modulation, Samples/Symbol
- Recovery of data
- Hex display helpful for observing patterns

Analyze recovered signal

- Automatic recovery of Preamble, Sync Word, Data, and any CRC/Checksum
- Again, not foolproof, but can manually assign labels





URH Process (1)

In this example of URH, We've started our analysis by performing a capture, saving it to disk and passing it to the analysis task with out URH startup screen. From there, using the Interpretation tab, we can preform some basic analysis, by changing our modulation, and samples per symbol in order to recover binary data. On the left-hand side, we see a signal captured with URH, with an FSK demodulation applied. This demodulation shows some repeating data highlighted in the waveform (at top) hex (in the middle) and the binary data (on the bottom). While not perfect, the "Autodetect Parameters" for automatic detection is incredibly handy for automatic recovery.

Next, moving to the Analysis tab, and the application of "Autodetect…" and a change of data output form binary to hex, we can begin to see repeating components of transmissions. URH will attempt to assign labels and highlight the appropriate sections of the transmission; we can add our own manually as well.

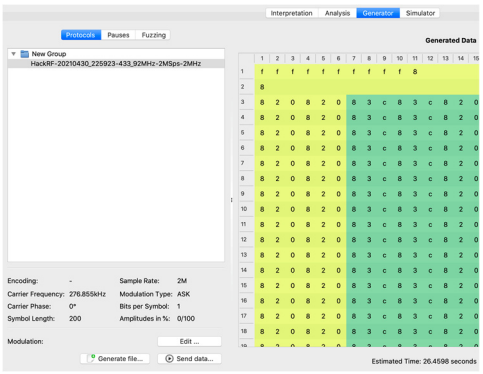
URH Process (2)

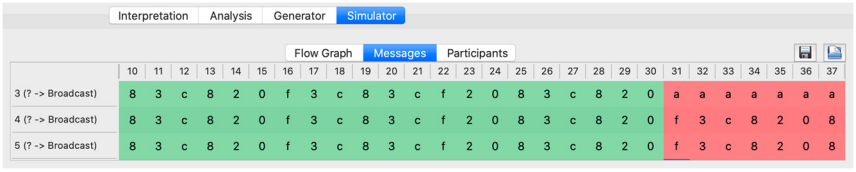
Replay unmodified with Generator

- Drag and Drop file to *Generated Data* and Send!

Modify all portions of transmissions with Simulator

- Drag and Drop file to *Generated Data*
- Edit in Messages
- Send!





URH Process (2)

Using URH and the Generator tab, we can highlight our capture in the left-hand group window and drag and drop some messages to "Generated Data" and send them unmodified with "Send Data…" for our unmodified replay attack.

If we wanted to manipulate the data, while retaining the same modulation and encoding as our original transmissions, we can use the Simulator tab. By dragging and dropping our messages to the Generated Data section, we can edit the messages in the Messages subtab, and inject them into the air with "Send data…".

Capture and Replay with Jeep Keyless Remote Entry

In 2016, Caleb Madrigal attacked his 2006 Jeep Patriot.

- Locked and *unlocked* his Jeep using replay-based attacks

Sit at car park, capture, replay open vehicles

- Can't start and drive away, but still scary!
- Unknown depth of issue for late-model cars and manufacturers

How do you update millions of vulnerable products that require a hands-on fix?

Have you put your laptop in the trunk of your car to keep it secure?

Capture and Replay with Jeep Keyless Remote Entry

In early 2016, Caleb Madrigal released a YouTube video and blog post (https://calebmadrigal.com/hackrf-replay-attack-jeep/) detailing his adventures using his HackRF One to perform simple replay-based attacks. Caleb hunted down the frequency (315 MHz) for the Remote Keyless Entry System for his 2006 Jeep Patriot and captured the signals with his HackRF One. Upon capture, he trimmed the relevant signals out of the large capture to the appropriate portions and subsequently replayed them to perform the same actions as those captured. Caleb did not release his code that he used to perform the attack, but it is easily re-creatable from the GCR flow graphs that he shared.

This poses an interesting attack scenario for this particular replay operation. In this case, an attacker can sit in a car park, observe lock and unlock operations, and capture these for later. This doesn't pose a significant risk for vehicle theft, in that this performs actions only against door unlocks and not any interaction with the operation of the vehicle. What happens when IT assets are stored in the vehicle in a public place or overnight at a residence where this can happen silently?

Ultimately, the lesson to be learned out of replay attacks in this scenario is that of resolution after time has passed. In this case, we are talking about nearly 10-year-old technology that was not prepared for the future. It is unknown how many vehicles (over several years) and manufacturers (that share in-vehicle subsystems) are resolving this problem. How do you go about patching and modifying the protocol across potentially millions of vehicles and key fobs where hands-on activity is required for the fix?

DSC Security (1)

Google reveals that they are *Door and window open/close sensors.*

In use in ADT alarm systems

- ADT is one of the largest monitored burglar alarm systems in the US for business and residential properties.

DSC Security (1)

Remember those DSC contacts we discussed earlier with *rtl_433*? Turns out, with a quick Google search, we can determine that those devices are often installed as part of an ADT burglar alarm system for door and window open/close sensors.

We could surmise that these devices are used to trigger various states with ADT alarm systems. ADT is one of the largest residential and commercial monitored alarm systems in the United States. While this case study focuses on this example, it is likely that similar systems and technologies are used by other vendors all over the world.

DSC Security (2)

How does the alarm panel react to:

- Continuous replay of a close status and the window opens?
- Replay of an open status and the window is closed….at 2:30 AM, 10 days in a row?

Replay attacks can have technical *and human* consequences.

```
$ rtl_433
<...trimmed for brevity...>
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
time      : 2021-04-28 13:04:32
model     : DSC-Security id        : 2291244
closed    : 1            event     : 0            tamper    : 0            battery_ok: 1
xactivity : 0            xtamper1  : 0            xtamper2  : 0            exception : 0
esn       : 22f62c       status    : 195
status_hex: c3           Integrity : CRC
```


DSC Security (2)

As we've illustrated in this section, it is possible to obtain and analyze data from these sensors and perform unmodified replay-based attacks. Building a baseline, we can observe that the device out put only changes the status (and status_hex) when the door or window opens or closes, and that there are only two values. If we capture and replay these statuses, can we manipulate the alarm system? This raises some question from the attach surface:

What happens if we continuously replay, in rapid succession the status for a single sensor indicating it is closed? Does the system trigger an "overload" condition, or does it do nothing? If we continue to replay, and the panel is receiving continuous close events, and the window opens, does the panel react? Is there a timing for (immediate) status change back to close where the alarm won't trigger? Have we drowned out the open transmission with our close replay?

These become some great technical questions to answer and attempt to fix. More importantly how to humans react to manipulation of the system? What does the alarm owner do if the replay of the window open event is sent at 2:30 in the morning, with replay to override the real close event? While one application of this would likely be considered anomaly and ignored by the user. What happens if we conduct the same attack, setting off the alarm every day at 2:30 AM for many days in a row? At random times of day or night? It is likely that the responsible parties put little faith in the operation in the system and either choose to ignore it, or disable it altogether? Without testing, we'll never know.

Summary

SDR platform selection is complex with many to choose from.

IoT security and operation can be affected with exposure to new spectrum by removing security through obscurity.

Capture and Replay simple and effective in many cases

Analysis of unknown signals is complex

URH can make the task easier

GNURadio can unlock all secrets, with considerable work

Summary

In conclusion for this section, we examined several core concepts for using SDR technology for the discovery of previously inaccessible protocols. We explored the SDR marketplace for affordable and robust options as well as how we can begin using them to visualize the RF spectrum for the discovery of signals, including long-term observation.

We also found that a huge amount of work has already been done on projects to reveal protocols and data that pose some risk due to a flawed implementation. We also found that lessons can be learned from these projects and that they can be reimagined for other purposes, especially on a penetration test that reveals other data or otherwise shows risk in systems used by organizations.

Exercise: Conducting a Replay Attack on IoT

Using your lab book, complete the exercise, *Conducting a Replay Attack on IoT*.

This page intentionally left blank.

COURSE RESOURCES AND CONTACT INFORMATION

AUTHOR CONTACT

Larry Pesce
larry.pesce.556@gmail.com
Twitter: @haxorthematrix

James Leyte-Vidal
jameslvsec556@gmail.com
Twitter: @jamesleytevidal

Steven Walbroehl
steven.walbroehl@halborn.com
Twitter: @HalbornSteve

SANS INSTITUTE

11200 Rockville Pike
Suite 200
North Bethesda, MD 20852
301.654.SANS(7267)

PEN TESTING RESOURCES

pen-testing.sans.org
Twitter: @SANSPenTest

SANS EMAIL

GENERAL INQUIRIES: info@sans.org
REGISTRATION: registration@sans.org
TUITION: tuition@sans.org
PRESS/PR: press@sans.org

This page intentionally left blank.