Report 3: Exploring the Vulnerabilities of IoT Devices

Shai Levin

Foreword

In this report, we will discuss common vulnerabilities which are prevalent among Internet-of-Things (IoT) devices, and the tools and techniques one can stage to attack them. We will also discuss how these vulnerabilities might be addressed.

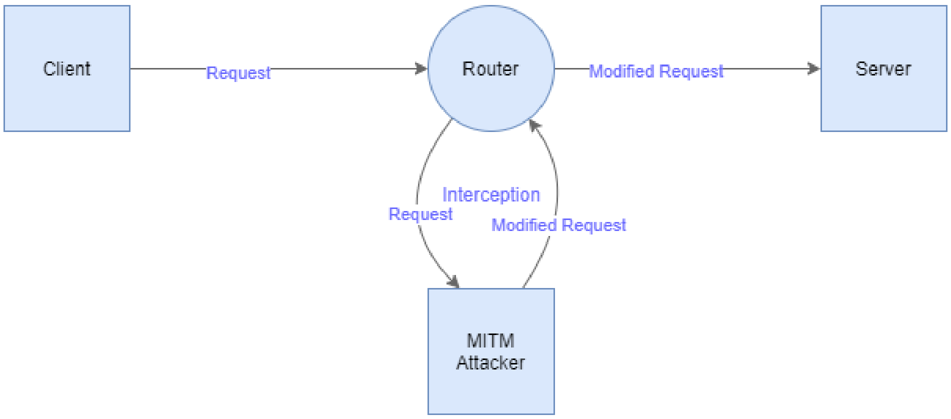
With the rise of the concept of Internet of Things, we see an increasing number of day to day objects being connected to the web. This new age of computing is still in its early ages, and as a consequence many security fundamentals are still lacking. Not unlike the early days of the internet when security practices were still in their infancy. The frequency of simple vulnerabilities being embedded in these smart devices is startling. Attackers might exploit these to gain the ability to control your lights, or unlock your door.

In this report we will take three experiments to demonstrate exactly this. In all cases users will be interacting with their Smart devices via their android mobile phone. In Section I we will crack a Bluetooth door lock, and open it. In Section II we will hijack a connection to a thermostat and adjust it's temperatures. In Section III we will do the same with a WiFi connected lightbulb, adjusting it's settings. In the TCP and UDP experiments, we assume the attacker has access to the wireless network, and may have cracked the wireless password similar to in Report 1. The Bluetooth attack has no prerequisites but will require extensive eavesdropping before one can gain access to the device.

Note most of these attacks employ the same fundamental concepts. They all involve either or both of these:

1. A man-in-the-middle (**MITM**) attack. This is when an attacker intercepts communications in a network between two parties, such that each party talks to the attacker believing they are talking to the other party. Also known as **Active Eavesdropping**.
2. **Passive Eavesdropping**. This is when an attacker listens to traffic over a wireless network and captures it for nefarious use later.

A diagram of a general **MITM** attack is depicted below, where an attacker pretends to be the server to the client, and pretends to be the client to the server. The attacker intercepts a request, modifies it, and sends it to the server.



Section I: TCP based devices

We start with TCP based Smart devices as they are by far the most prevalent today. TCP is frequently used as it guarantees reliable, in sequence communication. So what is the problem? Well, it turns out, many of these devices communicate over TCP, perhaps as HTTP or some proprietary application layer protocol. Frequently though, these protocols are unencrypted! This key flaw is what allows attackers to view transmissions between users' cellphone and their device in plaintext.

In this section we will analyse a wifi connected thermostat which is controlled by an app on a user's cellphone. The thermostat has several modes: heat, cool, fan and off. It operates at a variable temperature in fahrenheit. We will perform two experiments, one a passive eavesdropping attack, and one active attack. In the latter we will perform a TCP session hijack.

Tools Used

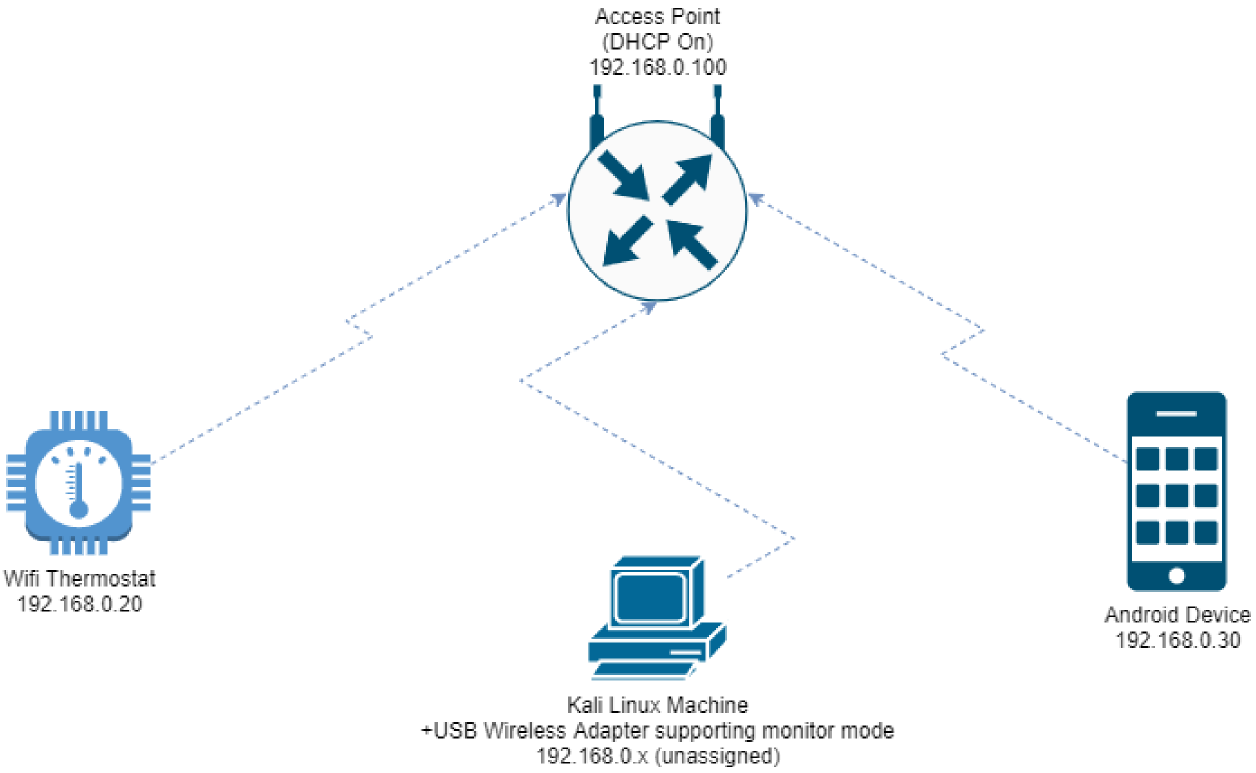
- Kali Linux OS
- WiPi or other mon mode supported wireless adapter
- airmon, wireshark, mitmproxy (for the active attack)

Kali and the software packages we use are free and easy to acquire. A mon mode supported wireless adapter is reasonably inexpensive and can be purchased for less than 50 NZD.

Passive Attack

We use the network configuration below, where DHCP is enabled and 3 devices are connected to the network:

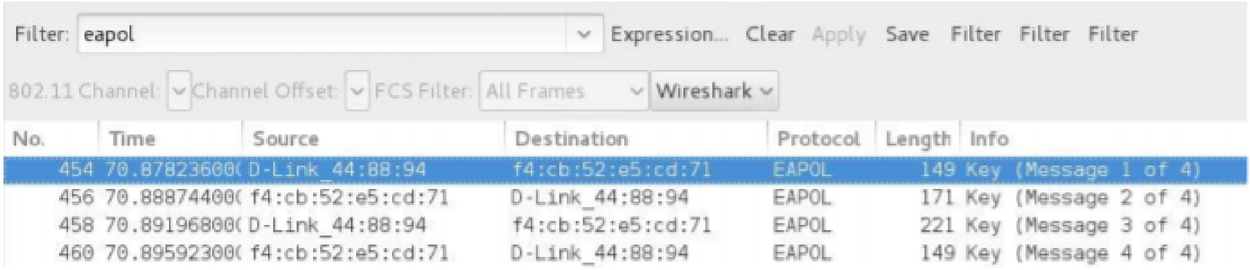
1. Wifi enabled thermostat
2. Good faith actor with android mobile
3. Attacker with a Kali Linux machine connected with a monitor mode enabled wireless device.



Now, we start by connecting the Kali machine to the wireless network. We kill the DHCP process on the Kali machine, so it is covertly listening to the wireless network without broadcasting it's IP address. As such it may be assigned but we do not care what it is.

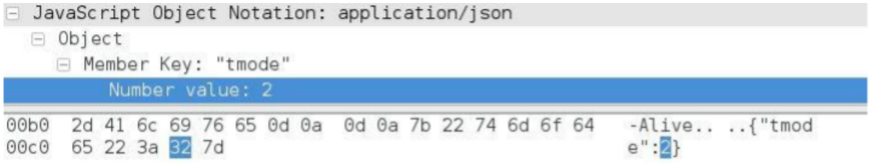
Next power on the thermostat. It should be configured to a static IP address on the access point. We then start the airmon-ng process to enable monitoring of the network and it's respective channel number. We connect wireshark to the mon0 interface, and configure decryption with the wireless password. As all wireless traffic is encrypted at the data link layer, We need to have this started before the android starts communicating in order to catch the EAPOL sequence so we can decrypt wireless communications between the devices.

We then connect the android device to the network. The EAPOL sequence is shown below via wireshark.

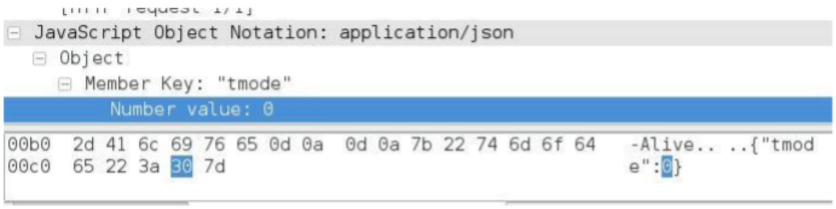

Filter: eapol
Expression... Clear Apply Save Filter Filter Filter
802.11 Channel: Channel Offset: FCS Filter: All Frames Wireshark

No.	Time	Source	Destination	Protocol	Length	Info
454	70.87823600(	D-Link_44:88:94	f4:cb:52:e5:cd:71	EAPOL	149	Key (Message 1 of 4)
456	70.88874400(	f4:cb:52:e5:cd:71	D-Link_44:88:94	EAPOL	171	Key (Message 2 of 4)
458	70.89196800(	D-Link_44:88:94	f4:cb:52:e5:cd:71	EAPOL	221	Key (Message 3 of 4)
460	70.89592300(	f4:cb:52:e5:cd:71	D-Link_44:88:94	EAPOL	149	Key (Message 4 of 4)

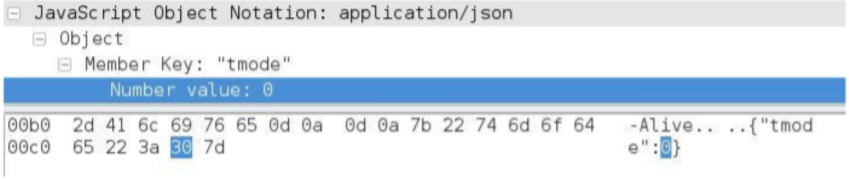
We can now operate the thermostat normally with the android app. Whenever an update is sent, the kali machine can capture the request in plaintext. Noting that the thermostat uses HTTP requests in plaintext with JSON payloads, which is a simple object type used commonly in web traffic. Below are some JSON requests captured via wireshark, demonstrating the user changing the operation mode. The attacker can use this information to learn how the protocol controls the thermostat.



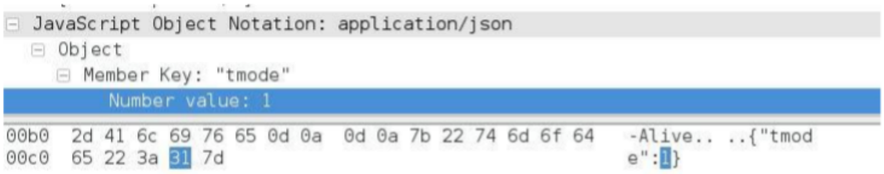
```
⊟ JavaScript Object Notation: application/json
  ⊟ Object
    ⊟ Member Key: "tmode"
        Number value: 2

00b0  2d 41 6c 69 76 65 0d 0a  0d 0a 7b 22 74 6d 6f 64   -Alive.. ..{"tmod
00c0  65 22 3a 32 7d                                      e":2}
```



```
⊟ JavaScript Object Notation: application/json
  ⊟ Object
    ⊟ Member Key: "tmode"
        Number value: 0

00b0  2d 41 6c 69 76 65 0d 0a  0d 0a 7b 22 74 6d 6f 64   -Alive.. ..{"tmod
00c0  65 22 3a 30 7d                                      e":0}
```



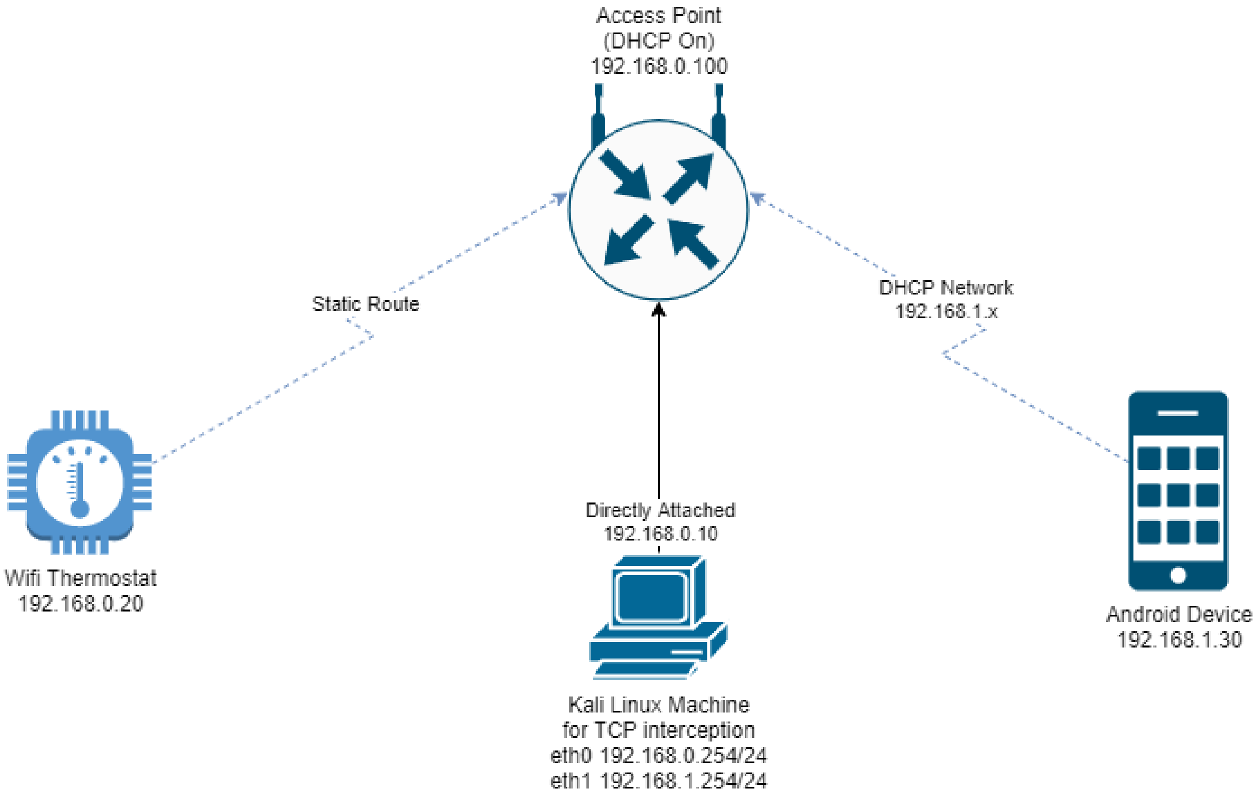
```
⊟ JavaScript Object Notation: application/json
  ⊟ Object
    ⊟ Member Key: "tmode"
        Number value: 0

00b0  2d 41 6c 69 76 65 0d 0a  0d 0a 7b 22 74 6d 6f 64   -Alive.. ..{"tmod
00c0  65 22 3a 30 7d                                      e":0}
```



```
⊟ JavaScript Object Notation: application/json
  ⊟ Object
    ⊟ Member Key: "tmode"
        Number value: 1

00b0  2d 41 6c 69 76 65 0d 0a  0d 0a 7b 22 74 6d 6f 64   -Alive.. ..{"tmod
00c0  65 22 3a 31 7d                                      e":1}
```

Active Attack

Next we use a similar network configuration to above, with some caveats. We need to introduce a redirection mechanism so that mitmproxy can be used. We have put the thermostat and android on two separate subnets so the router decides where to redirect traffic at the network layer. Typically, this configuration might take place when a firewall sits as a proxy on the router. We also configure our kali machine to redirect any TCP traffic from the 192.168.1.XX subnet, and enable IP forwarding. This is where mitmproxy comes in. It is a tool that acts as a proxy for the router, and controls where traffic should be routed on the network, by knowing how to talk to each platform's redirection mechanism. For the purpose of simplicity, we directly attach the Kali machine, but it can still operate as if connected wirelessly.



We set up the Kali Linux machine to have two ethernet interfaces. One which will talk to the thermostat, and one to the android device. We then start the mitmproxy process, and now the Kali machine acts as a router between 192.168.1.XX and 192.168.0.XX networks. We demonstrated several scripts in the lab which will modify the android devices requests whenever they change the temperature on the app. Here are some of the functions we tested.

1. Changing the temperature by a factor or +-10F
2. Swapping Heat and Cool modes

3. Forcing the thermostat on a specific temperature and mode, whilst still displaying a false reading on the android app. This means the android user might be fooled into thinking the device is operating normally, but is in fact being controlled by the attacker.

Option 3 is the most interesting, and demonstrates a nefarious strategy. A user might not even realise they are being sabotaged. Consider if this thermostat controlled a critical infrastructure, such as a hot water cylinder or product freezer. This could cause huge losses to a business or a safety hazard. This bears resemblance to many ICT hacks done over the last several years, such as Stuxnet. An unprecedented attack that effectively disabled the Iranian nuclear program.

Section II: UDP based devices

UDP based IoT devices will operate over a user's wireless network. Typically a device will use UDP rather than TCP when it requires more fast, low latency updating, as UDP is faster and simpler in design to TCP. However, UDP is severely lacking in aspects such as reliability, and security. UDP based attacks are in some ways easier, and in some ways harder than TCP. It is harder to intercept packets as with TCP with a MITM proxy as UDP packets are connectionless, meaning they are ‘fast moving’ within a network. It might not be possible for the attacker to intercept all packets between the user and the device. However, it is easier in the sense that UDP packets do not guarantee their origin, so it is possible to spoof UDP packets and send them to the device at a later date. In this lab we will demonstrate an interception type attack. However, in the UDP case, it is likely much easier to simply capture packets, modify them, and send them to the device at a later time, rather than trying to directly intercept them.

We will demonstrate an attack against a Lifx Smart bulb using UDP connection. The Lifx Smart bulb is controlled via a users android app, over a wireless network. The user can adjust settings such as colour, intensity and white colour temperature.

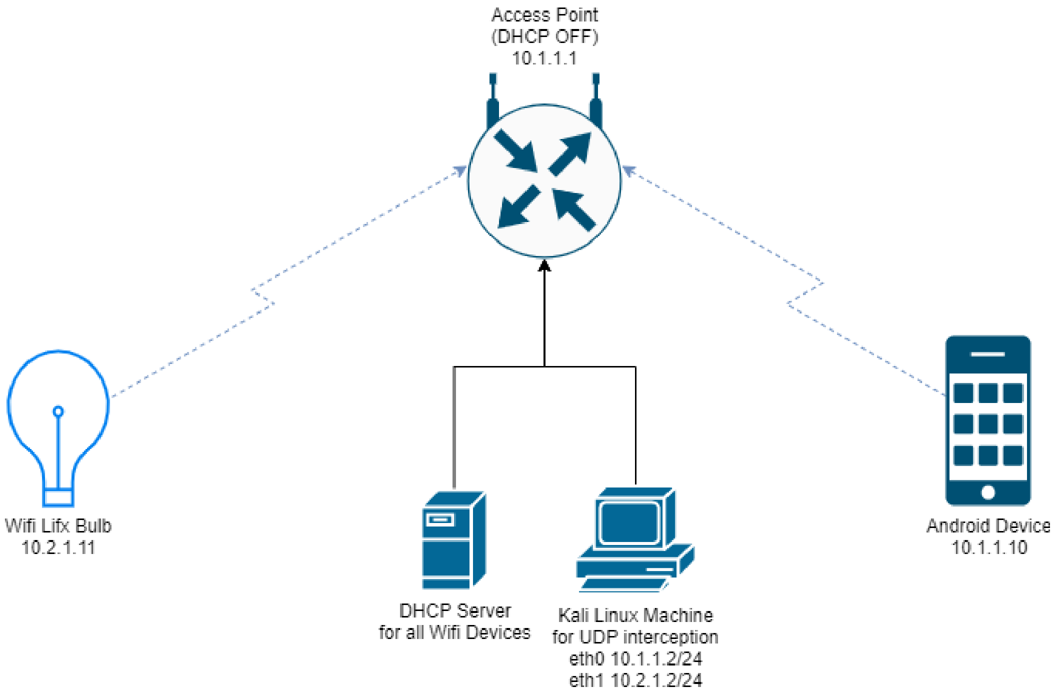
Tools Used

- Kali Linux Machine connected via ethernet (but could equally be done with any wireless adapter provided the attacker knows the password)
- Hexinject

It is quite critical that the Kali machine has a fast stable connection, as it needs to ‘catch’ the UDP packets before they are sent to the light bulb.

Attack

We use the network configuration described below, where DHCP is handled at a back-end server on the network. This is similar to our TCP experiment 2, where we need a point that the Kali machine can hijack the routing of traffic.



We configure IP forwarding on the Kali machine so that IP packets are forwarded across two interfaces. We configure two interfaces for each subnet. Wireshark was also enabled so we could capture the UDP traffic. Now, the attack is quite simple. Whenever the android device attempts to transmit an update to the bulb, we use hexinject to intercept these UDP packets and modify them to our desired effect. We tried several modifications of these packets, namely:

1. When the bulb is turned on, the interception proxy modifies the packet to turn the bulb off.
2. When a colour is altered, it is intercepted to change it to a fixed colour.
3. When intensity is changed to 100%, it is intercepted to turn the bulb off.

Again, it is likely much easier to simply capture these packets using wireshark, modify them, and send them to the bulb at a later point. This would circumvent the issue of UDP packets getting past the MITM proxy.

So what is the issue here? Well, it is twofold. Firstly, the UDP packets are completely unencrypted and unauthenticated. Because it is **unencrypted**, an attacker can easily read what the traffic is and modify it to their needs. Because it is **unauthenticated**, the attacker can easily pose as the android device, and issue commands to the bulb.

Section III: Bluetooth 4.0 based devices

Bluetooth pairing is another extremely common feature of IoT devices. With several modes of authentication in Bluetooth 4.0, there poses a risk of eavesdropping attacks. The 4 modes are as follows, using a temporal key to negotiate pairing:
Just Works(™): The temporal key used for authentication is set to 0.
Passkey: The temporal key is a random 6 digit number passed between the device.
Out of Band pairing (OOB): A more secure version which uses a 128 bit temporal key and a separate channel designed to be immune to BL eavesdropping.

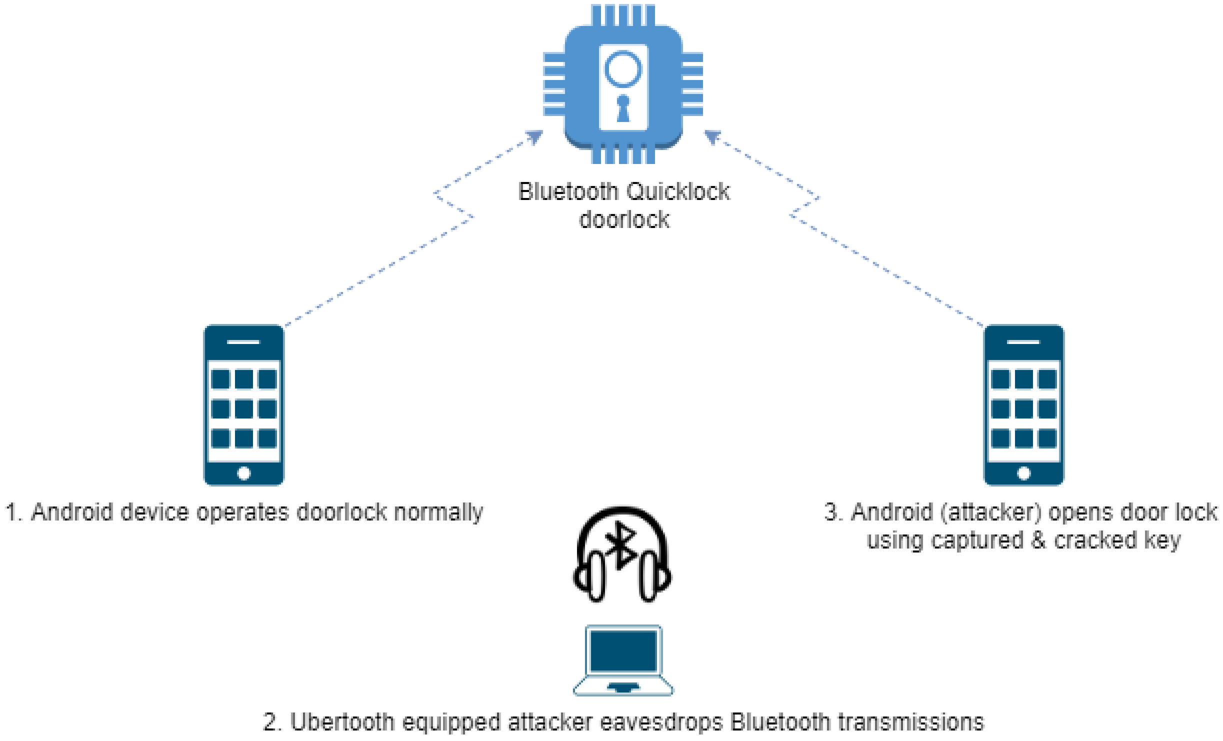
If an attacker eavesdrops these pairing packets of the first two modes, it can very quickly brute force crack the temporal key, and use this to derive the long term key. Bluetooth is an extremely noisy protocol. There is lots of traffic in the air. However, if an attacker had persistence and time, they could eventually capture a full handshake while being 10-20m away from the device.

In this experiment we crack the key of a Bluetooth Quicklock doorlock. This Bluetooth 4.0 device uses the **Just Works(™)** mode of authentication.

Tools

- Ubertooth device
- Kali linux machine
- Crackle software package

The ubertooth is effectively a bluetooth monitoring device, which can listen to nearby bluetooth traffic for capturing. Crackle is the tool we use to crack the bluetooth temporal key.

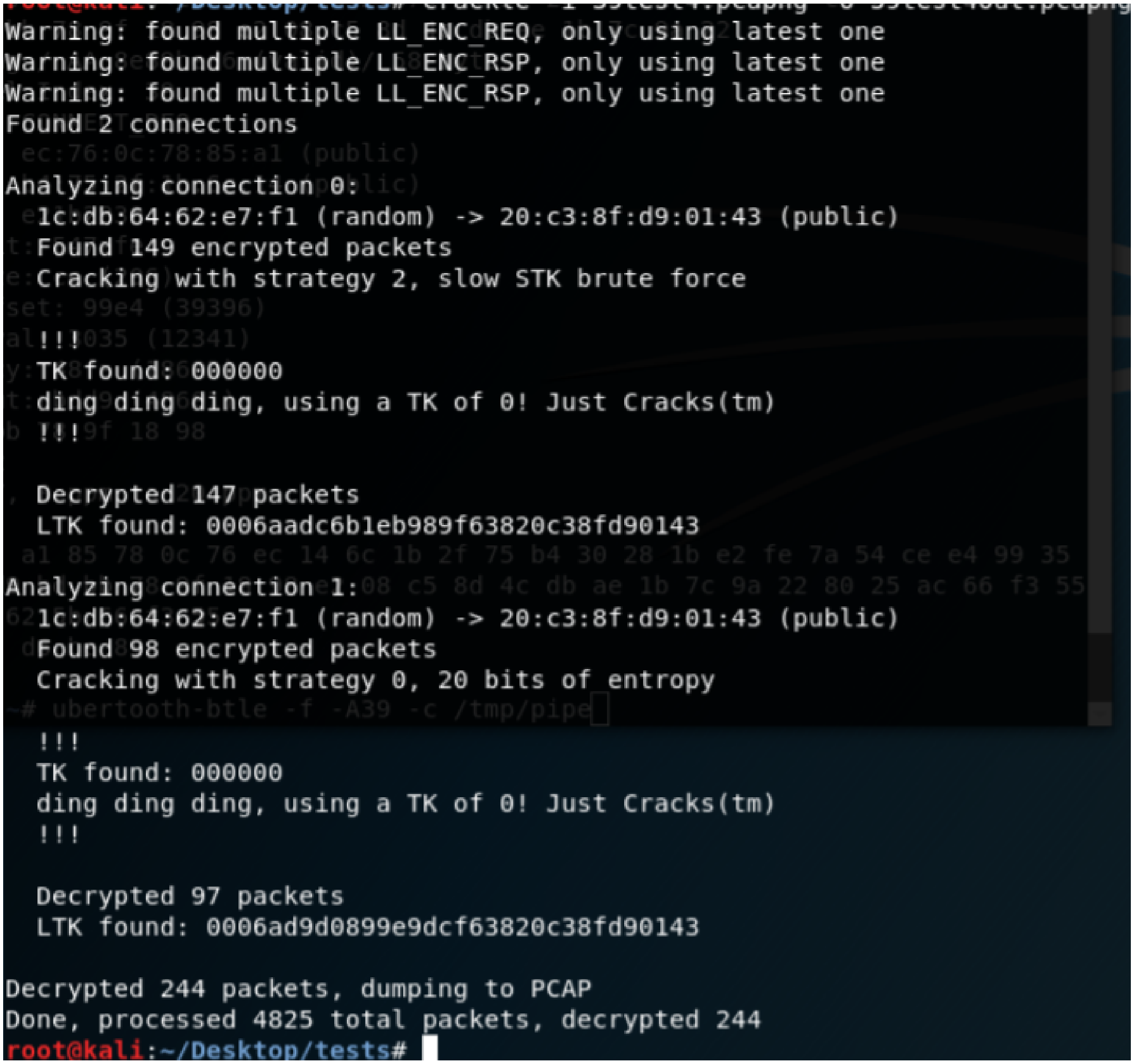

Bluetooth Quicklock doorlock
1. Android device operates doorlock normally
3. Android (attacker) opens door lock using captured & cracked key
2. Ubertooth equipped attacker eavesdrops Bluetooth transmissions

Attack

The process is simplified in the diagram above. Before we begin the lock is set to a pin that is connected to the SafeTech app on the first android device, but we unpair and forget the device in the bluetooth settings. We configure our Ubertooth to pipe all bluetooth traffic into a .pcap file on our Kali machine which we can inspect in wireshark.

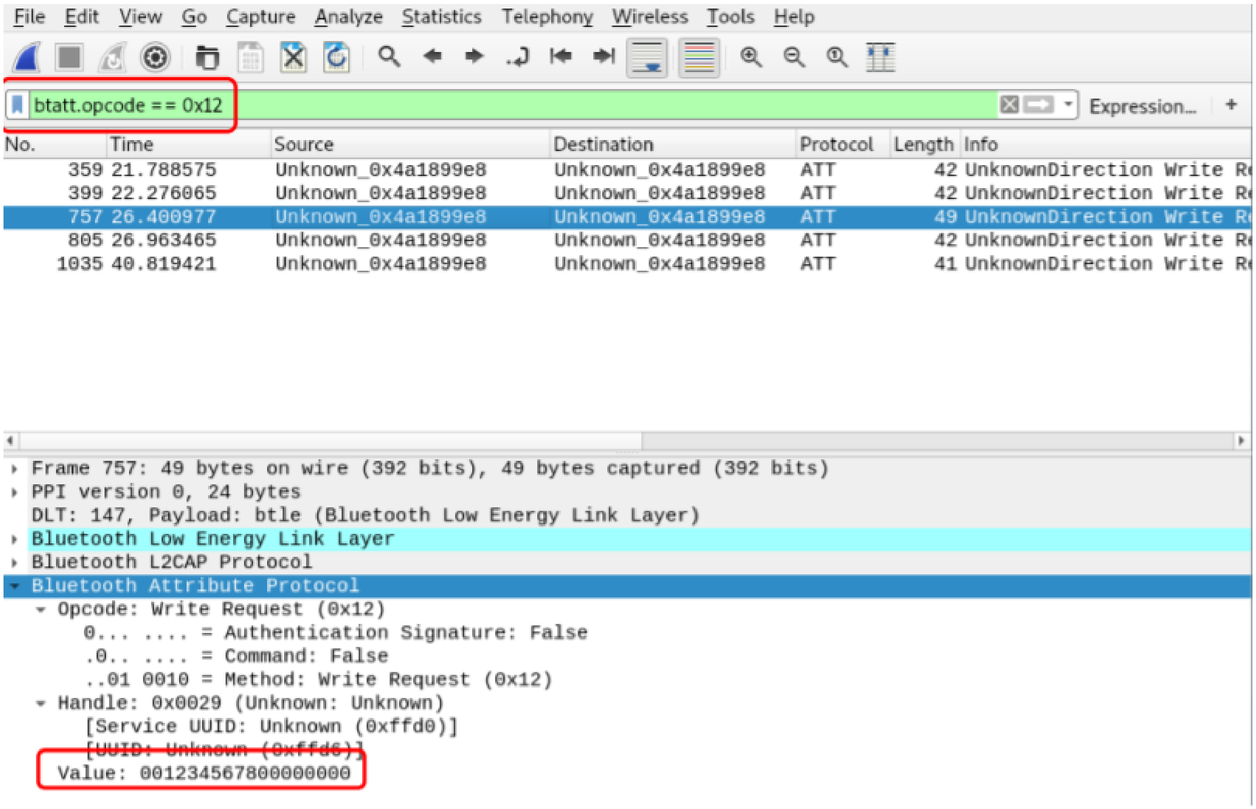
Now we start the ubertooth capture and immediately begin the pairing process. We press the pairing button on the lock, open the SafeTech app on the device and connect. At this point we should have captured the handshake. We may need to repeat this process, but in practice this capturing will be streamlined.

We then use crackle to decrypt the packets in our .pcap file. If we have captured the handshake successfully, we get the following:



```
Warning: found multiple LL_ENC_REQ, only using latest one
Warning: found multiple LL_ENC_RSP, only using latest one
Warning: found multiple LL_ENC_RSP, only using latest one
Found 2 connections

Analyzing connection 0:
  1c:db:64:62:e7:f1 (random) -> 20:c3:8f:d9:01:43 (public)
  Found 149 encrypted packets
  Cracking with strategy 2, slow STK brute force

  !!!
  TK found: 000000
  ding ding ding, using a TK of 0! Just Cracks(tm)
  !!!

  Decrypted 147 packets
  LTK found: 0006aadc6b1eb989f63820c38fd90143

Analyzing connection 1:
  1c:db:64:62:e7:f1 (random) -> 20:c3:8f:d9:01:43 (public)
  Found 98 encrypted packets
  Cracking with strategy 0, 20 bits of entropy

  !!!
  TK found: 000000
  ding ding ding, using a TK of 0! Just Cracks(tm)
  !!!

  Decrypted 97 packets
  LTK found: 0006ad9d0899e9dcf63820c38fd90143

Decrypted 244 packets, dumping to PCAP
Done, processed 4825 total packets, decrypted 244
root@kali:~/Desktop/tests# 
```

Now we can open the output of crackle decryption in wireshark. We are looking for a value somewhere in the Bluetooth Attribute Protocol (btatt.opcode == 0x12).


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
btatt.opcode == 0x12
Expression...
+

```
No.     Time        Source                Destination           Protocol  Length Info
    359 21.788575   Unknown_0x4a1899e8    Unknown_0x4a1899e8    ATT           42 UnknownDirection Write R
    399 22.276065   Unknown_0x4a1899e8    Unknown_0x4a1899e8    ATT           42 UnknownDirection Write R
    757 26.400977   Unknown_0x4a1899e8    Unknown_0x4a1899e8    ATT           49 UnknownDirection Write R
    805 26.963465   Unknown_0x4a1899e8    Unknown_0x4a1899e8    ATT           42 UnknownDirection Write R
   1035 40.819421   Unknown_0x4a1899e8    Unknown_0x4a1899e8    ATT           41 UnknownDirection Write R
```

```
▸ Frame 757: 49 bytes on wire (392 bits), 49 bytes captured (392 bits)
▸ PPI version 0, 24 bytes
  DLT: 147, Payload: btle (Bluetooth Low Energy Link Layer)
▸ Bluetooth Low Energy Link Layer
▸ Bluetooth L2CAP Protocol
▾ Bluetooth Attribute Protocol
  ▾ Opcode: Write Request (0x12)
      0... .... = Authentication Signature: False
      .0.. .... = Command: False
      ..01 0010 = Method: Write Request (0x12)
  ▾ Handle: 0x0029 (Unknown: Unknown)
      [Service UUID: Unknown (0xffd0)]
      [UUID: Unknown (0xffd6)]
    Value: 00123456780000000
```

Here we have the value of the decrypted password - 01234567 - padded with zeroes. Now, we use a second android device and try to pair the device with the SafeTech app. We enter the code and unlock the door.

As we can see here, a door lock device should not be using a temporal key of 0, nor even a 6 digit pin - as that can still easily be cracked. The more secure mode of Bluetooth 4.0 authentication, Out of Band Pairing (OOB), might prevent this attack. The issue with OOB is that it requires additional circuitry and manufacturers will be unwilling to add this as it increases cost. Another solution would be using a Bluetooth 4.2 or 5.0 supported device, but again, this suffers from compatibility and cost issues.

Conclusion

In conclusion, these are the primary issues facing IoT device security:

98% of IoT device traffic is unencrypted
According to a report by Unit 42, this shocking statistic shows how most IoT devices are vulnerable to eavesdropping attacks similar to the ones we posed in this lab.

No separation of IoT assets and IT assets
Businesses might not be aware of the risks posed when a secure network containing confidential IT assets is integrated with IoT devices. Once an attacker gains access to a network, they may use these IoT devices to backdoor themselves into secure machines, move laterally across the network, or gather intelligence about a business.

Cheap devices means cheap security
Consumers want the cheapest device to suit their needs. To save costs, cheap devices will have weaker computing power and thus their security capability is constrained. The common issue we saw with all the devices in this report was that they were made with low cost in mind, and the security vulnerabilities they present reflect this.

How might one address these problems in their business environment?

1. **Invest in quality IoT devices with a trusted pedigree.**
   The added cost of a device with secure authentication protocols is well worth it in a business environment.
2. **Isolate IoT communications within a business network**
   This minimises the risk of IoT vulnerabilities within your network.
3. **Educate yourself on the vulnerabilities of IoT devices.**
   If managers and engineers are aware of the risks of using smart devices, they can make more informed decisions on their strategy.

References:

- More on Bluetooth security and which options are eavesdrop resistant.
  https://forum.digikey.com/t/a-basic-introduction-to-ble-4-x-security/12501
- Why 98% of IoT traffic is unencrypted
  https://www.iot-now.com/2020/06/04/103245-why-98-of-iot-traffic-is-unencrypted/
- Report on IoT security
  https://unit42.paloaltonetworks.com/iot-threat-report-2020/
- Diagrams made using draw.io