



探索低功耗蓝牙之看我如何逆向物联网灯泡并控制它

Covfefe

2017-08-31 共326486人围观，发现 5 个不明物体

极客

终端安全

作为一个IoT发烧友和夜猫子安全研究员，IoT一直吸引着我，因为我们每天使用的IoT应用程序都在让生活变得更加轻松便利。从使用一个应用程序到监视，控制设备，这是我一直以来想深入的部分，而这一切都成为可能。

我最近在亚马逊上为我的书桌买了一个智能LED灯泡，这是一个7瓦的[Syska Smartlight Rainbow LED](#)。还可以通过应用程序，在安卓移动设备上用蓝牙来控制，玩这个灯的时候发现它很有趣，如果whatsapp新的通知，这个灯就会闪烁。它还能在早上叫醒我，总之我可以用很多方式和它进行交互，但是只能从自带的应用程序上操作。

不像市场上其他著名的智能灯泡（比如Philips Hue，LIFX），这款灯泡由印度的一家公司生产，它使用低功耗蓝牙（BLE）而不是Wifi，而且没有API来与定制应用程序交互，我当时还是决定买了它，因为价格便宜（约135人民币）。

在用了几个星期后，我决定看看这个外壳底下是什么样的，我研究蓝牙和低功耗蓝牙协议已经有一段时间了，所以对这方面有所了解（这得感谢Cypress Semiconductor去年送了我一个PSoC4 BLE 开发板）。BLE（低功耗蓝牙，Bluetooth Low energy的缩写）基本上在通信层上提供了进行用户定义服务的方法。为生产商在他们正在制造的设备上定义特有的协议配置文件，尽管BLE协议已经有一些已经定义好的配置文件，比如旧版的UART，BLE心率监测器等等，生产商还是可以自由使用所谓的GATT，Generic Attribute Profile，然后创建他们自己的自定义配置文件，这些配置文件决定了它们要如何在主设备和从设备之间通信。

事实上，这个灯泡没有使用基于TCP/IP协议的通信，这就使得逆向变得有点困难了，我的意思是：如果这个灯是在家庭网络中，事情不就变得更加容易了吗？我可以使用Wireshark通过它的MAC或IP来嗅探数据包，然后保存在PCAP文件中方便之后的分析，数据包是很容易嗅探的，基本上类似于中间人这种东西，甚至一个简单的CLI tcpdump也有用，但是现实是这是使用**对等网络**的蓝牙，在同一时间内，一个BLE设备能与一个中心设备通信。

从之前的BLE项目中我了解到Nordic Semiconductor的一款在安卓或IOS上运行的一个神奇的应用程序，它叫[nrf connect](#)，可以用来探索设备暴露出的GATT service和characteristic，我可以用它来连接我的灯泡，然后就能知道它的唯一地址（某种MAC地址）还能找到GATT service和响应的characteristic，这是

很好的起点不是吗？我迅速在我的安卓手机上安装并打开了nrf connect，打开蓝牙并扫描设备，结果迅出现在屏幕上，名叫“Cnligh”。

Devices

SCAN

SCANNER BONDED ADVERTISER

No filter

Cnligh
88:C2:55:CA:F0:36
BONDED -80 dBm ↔ 34 ms

CONNECT

Type: LE only
Flags: GeneralDiscoverable, BrEdrNotSupported
Incomplete List of 16-bit Service UUIDs: 0xF371
Complete Local Name: Cnligh

CLONE RAW MORE



连接之后显示三个GATT service，如下图所示：

The screenshot displays a mobile application interface for managing Bluetooth connections. At the top, there's a header bar with the title "Devices" and a "DISCONNECT" button. Below this, a list of devices is shown:

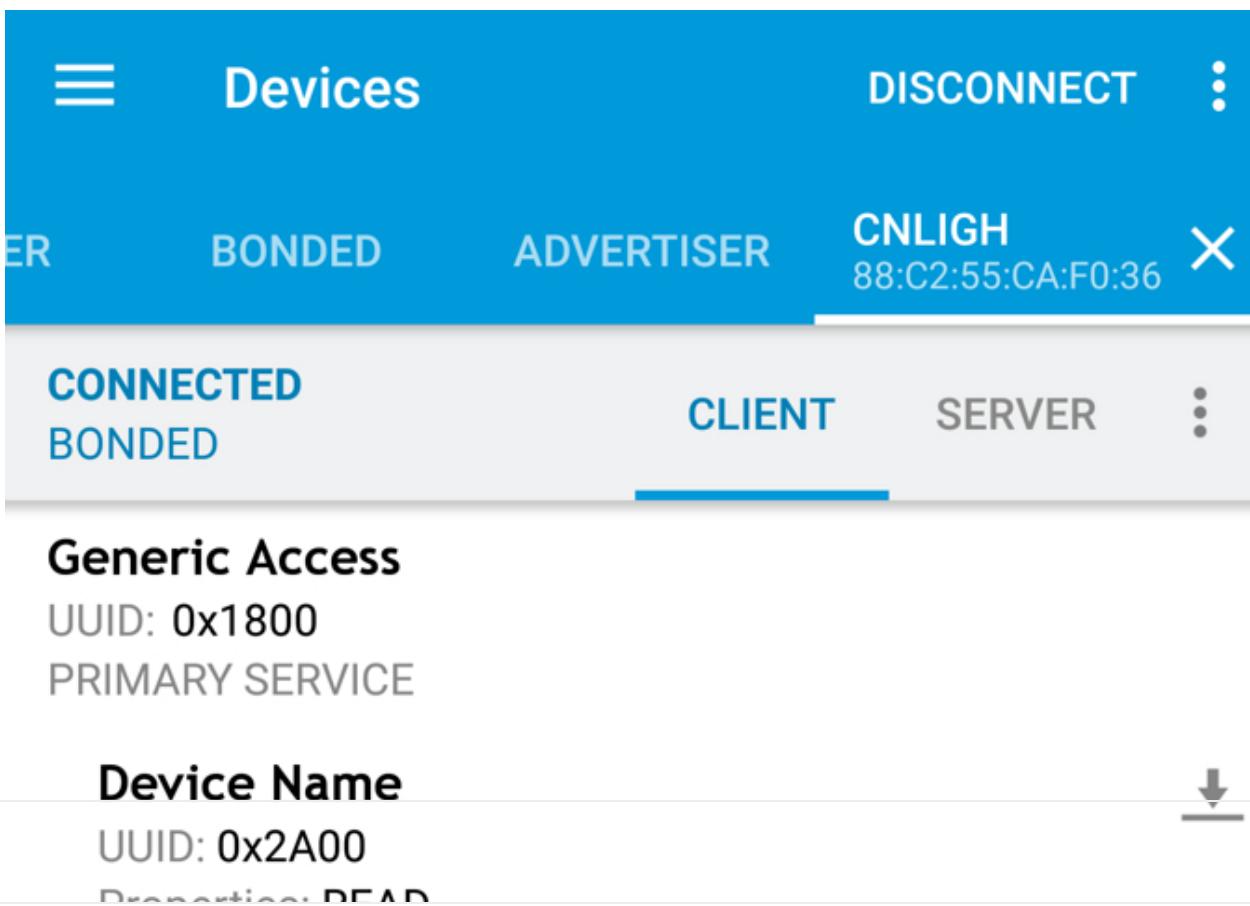
ER	BONDED	ADVERTISER	CNLIGH 88:C2:55:CA:F0:36
CONNECTED			CLIENT
BONDED			SERVER

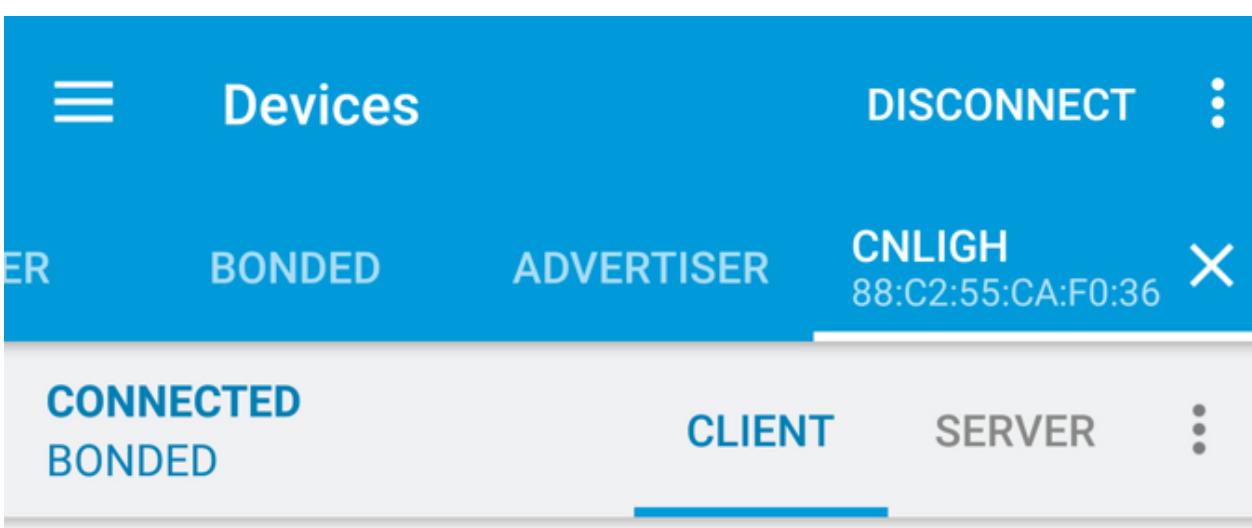
Below the device list, three GATT services are listed:

- Generic Access**
UUID: 0x1800
PRIMARY SERVICE
- Generic Attribute**
UUID: 0x1801
PRIMARY SERVICE
- Unknown Service**
UUID: 0000f371-0000-1000-8000-00805f9b34fb
PRIMARY SERVICE



灯泡暴露的三个GATT service中，有两个在大多数BLE设备上通用，它们分别是UUID为0x1800的Generic Access和UUID为0x1801的Generic Attribute，你可以查看[Bluetooth GATT website](#)来查找更多，这两个service定义了设备名，设备种类和状态，如下图所示：



Properties: READ	
Appearance	
UUID: 0x2A01	
Properties: READ	
Peripheral Privacy Flag	
UUID: 0x2A02	
Properties: READ, WRITE	
Reconnection Address	
UUID: 0x2A03	
Properties: WRITE	
Peripheral Preferred Connection Parameters	
UUID: 0x2A04	
Properties: READ	
Generic Attribute	
UUID: 0x1801	
 <p>The screenshot shows a mobile application interface for managing Bluetooth connections. At the top, there's a header with a menu icon, the word 'Devices', a 'DISCONNECT' button, and a more options icon. Below this, a blue bar displays connection status: 'BONDED' for one device and 'ADVERTISER' for another, with the MAC address '88:C2:55:CA:F0:36' and name 'CNLIGH'. A red 'X' button is next to the MAC address. The main list shows a single entry: 'CONNECTED BONDED' for the same device, with 'CLIENT' and 'SERVER' roles indicated. A blue progress bar is at the bottom of this list. At the very bottom, there's a section for 'Generic Access' with a 'UUID: 0x1800' label.</p>	
Generic Access	
UUID: 0x1800	

PRIMARY SERVICE

Generic Attribute

UUID: 0x1801

PRIMARY SERVICE



Service Changed

UUID: 0x2A05

Properties: INDICATE

Descriptors:

Client Characteristic Configuration



UUID: 0x2902

Unknown Service

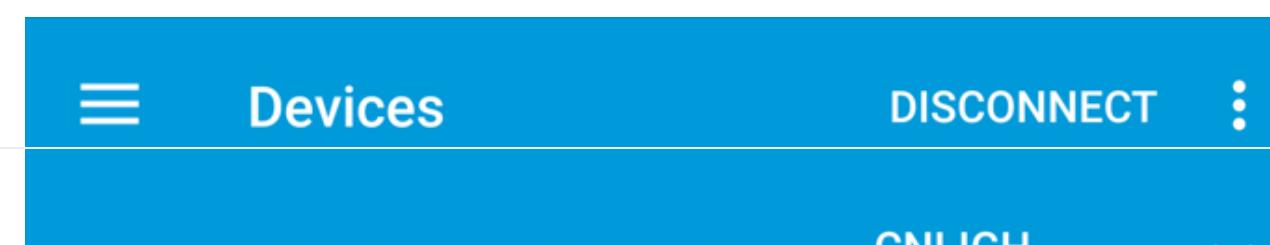
UUID: 0000f371-0000-1000-8000-00805f9b34fb

PRIMARY SERVICE



Wireless by Nordic

除了那两个，还有一个是用户定义的service，看到128位的UUID，我可以说这是生产商定义的BLE GATT profile。



ER BONDED ADVERTISER ONLET 88:C2:55:CA:F0:36 X

CONNECTED **CLIENT** SERVER :

UUID: 0000f371-0000-1000-8000-00805f9b34fb
PRIMARY SERVICE

Unknown Characteristic  

UUID: 0000fff1-0000-1000-8000-00805f9b34fb
Properties: READ, WRITE

Descriptors:

Characteristic User Description  

UUID: 0x2901

Unknown Characteristic  

UUID: 0000fff2-0000-1000-8000-00805f9b34fb
Properties: READ, WRITE

Descriptors:

Characteristic User Description  

UUID: 0x2901

Unknown Characteristic  

UUID: 0000fff3-0000-1000-8000-00805f9b34fb
Properties: READ, WRITE

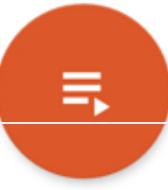
Descriptors:

Characteristic User Description  

UUID: 0x2901

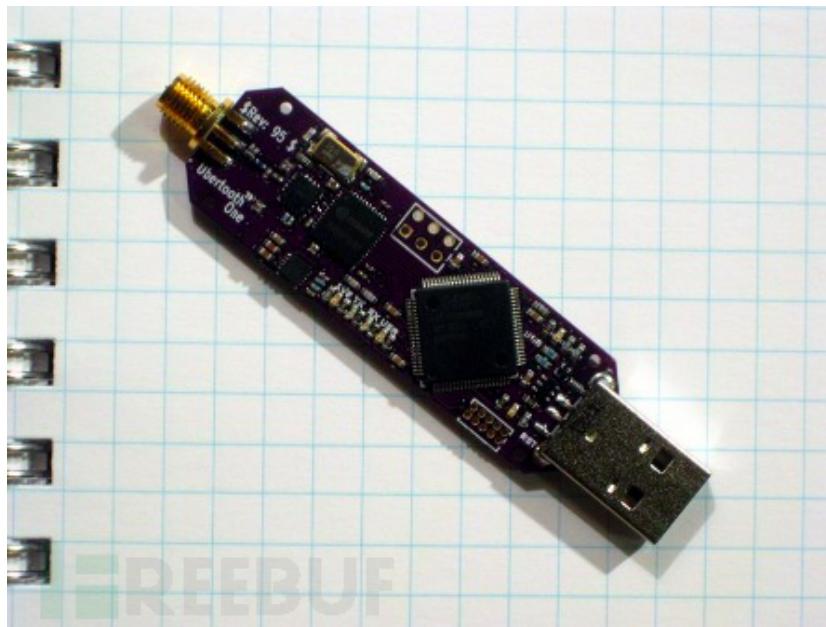
Unknown Characteristic  

UUID: 0000fff4-0000-1000-8000-00805f9b34fb
Properties: READ, WRITE



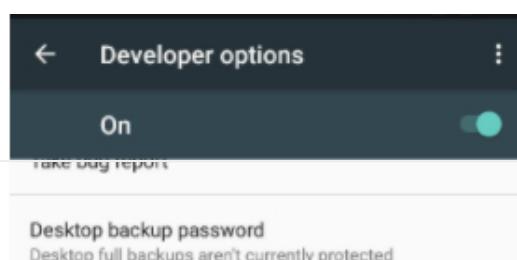
现在看service，我当时很确定其中一个characteristic就是控制灯泡颜色的，但是之后我非常失望，因为当时期待能有个叫RGB color或者其他什么的，但是所有service都被标记为unknown characteristic，但是我找到了service的名字，UUID，我就能发送一些数据包看看是什么反应，但事实并非如此，因此下一步能嗅探BLE应用程序发给灯泡的数据包，我在这次的搜索中知道了设备地址，GATT service和读取的或写的characteristic，这对之后的步骤是很有帮助的。

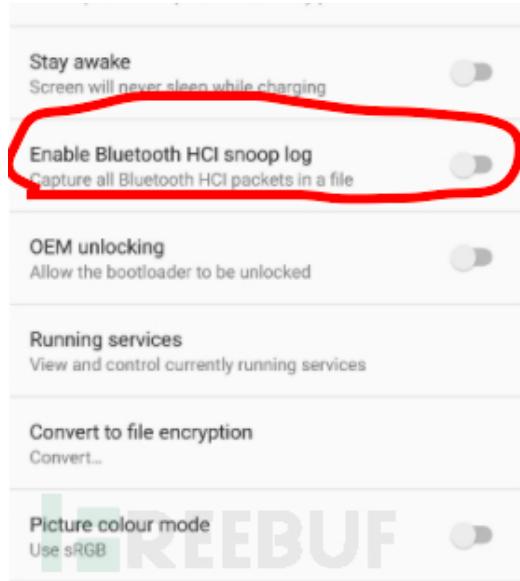
我听说过Great Scott的一个叫Ubertooth One的设备，它能嗅探蓝牙数据包，但是看到价格和产地，我决定寻找替代方案，经过查找，我发现还有其他Nordic和Cypress Semiconductor的一些硬件，不过研究一个灯泡，却花费多于一个灯泡的钱，这没有意义不是吗？



进一步的谷歌搜索之后，我在StackOverflow上找到了替代方法，Android 4.4 (katkat) 可以将蓝牙数据包记录在一个文件中，我深入了解了一下，发现需要启用开发者模式，连接蓝牙设备然后交互，所有记录会被记录到SD卡中的“btsnoop_hci”文件中。

如果你使用的是安卓4.4 (kitkat) 或更高版本的手机，那么就可以通过设置 – 开发人员选项 – 启用蓝牙监听日志来启用此功能，这是蓝牙调试工具的基础，启用之后，所有蓝牙日志会被记录到文件“btsnoop_hci”中，我启用了这个功能，并且运行了控制灯泡的应用程序，和往常一样改变灯泡的颜色，这次我更多地注意在基本颜色上，比如红色，蓝色和绿色，这将帮助我在分析包时过滤数据流。关掉程序，文件果然生成了！一个不到20Kb的日志文件就能控制灯泡，而不需要拆开它:P

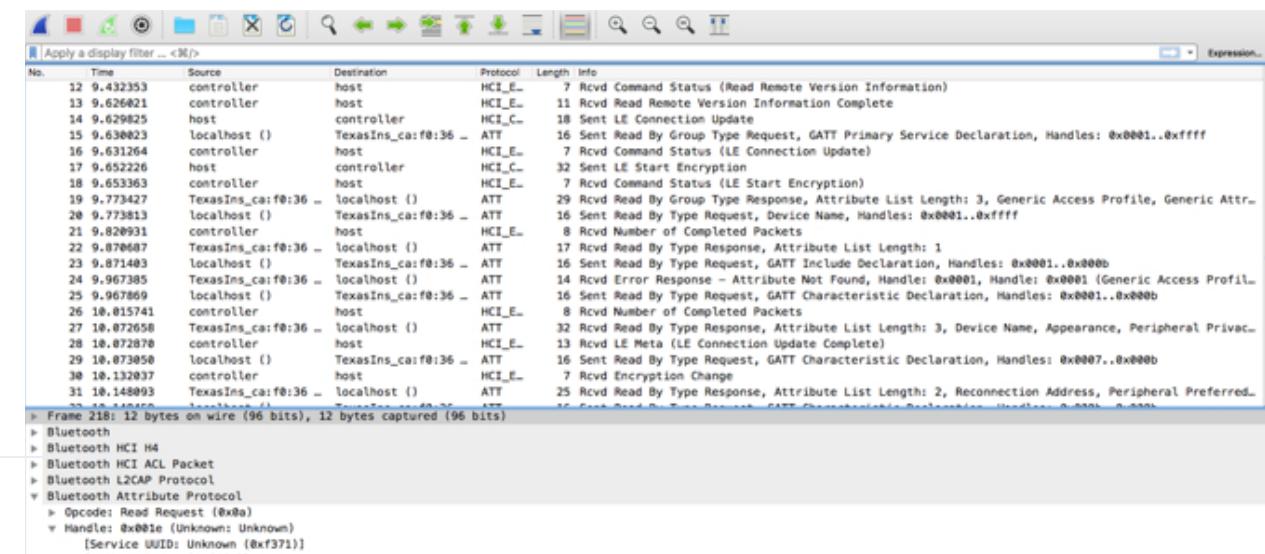




下一步就是将文件拷贝到电脑上，用Wireshark将其可视化，说真的，Wireshark真是个超级棒的工具，过了解基本的教程，然后动手实践，就能找到很多有趣的东西。

灯泡工作的方式可能是通过从应用程序中获取红，绿，蓝或所谓的RGB值，然后通过更改灯泡中相应的颜色来响应，也有可能直接使用颜色名称，然后通过EEPROM中的查找表来映射强度，从几个过去的RC LED项目中，我知道任何颜色可以都映射为红色绿色和蓝色强度的值，通常是8位（0-255），0表示关闭，255表示来自RGB的特定颜色的全部强度，这些强度使用来自定时器IC或微控制器的脉宽调制信号（PWM）来改变。我过去已经做过这样的项目，如果有兴趣去查看我的项目[IoT Holicay Lights](#)和[OpenHAB RGB](#)控制器，这些使用的都是相同的概念。

如果仔细看下面的截图，你会发现一些有趣的事情，destination有两个标签/值：localhost，也就是我的安卓手机，另一种是“Texas Instruments”，具有特定的唯一地址，谷歌了一下，我发现它是Texas Instruments的基于BLE的芯片“CC2540”，而灯泡用的可能就是这个，我们之前从GATT service收集UUID也表明是相同的芯片:D 所以现在就算不打开灯泡，我们也知道里面是什么了;)





进一步研究之后，我发现整体通信中涉及的协议种类较少，在Wireshark中可以看到：HCl_E, HCl_C, ATT等等，ATT看起来很有趣，应用ATT过滤器可以只显示ATT数据包，然后我应用了蓝牙逻辑链路控制和适配协议(bt2cap) 的过滤器，并尝试分析从本地主机发送到灯泡的数据包。

研究了20-40个数据包之后，我发现了轻微变化的字符串，看下面：

```
Value: 00100006000a03000101000025ff00000000
Value: 00110006000a030001010049ff0000000000
Value: 00120006000a0300010100ff000000000000
Value: 00130006000a030001010049ff0000000000
Value: 00140006000a03000101000025ff00000000
```

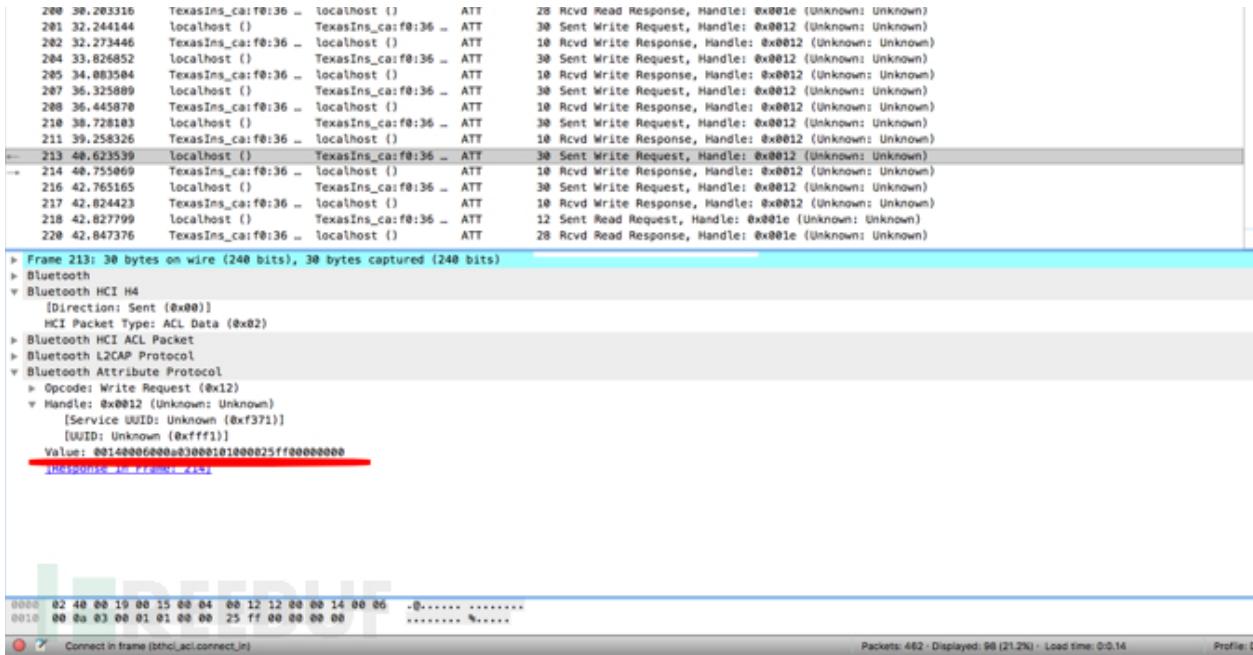
发现了吗？再弄个再简单一点的：

```
Value: 0010000 6000a0300010100 0025ff 00000000
Value: 0011000 6000a0300010100 49ff00 00000000
Value: 0012000 6000a0300010100 ff0000 00000000
Value: 0013000 6000a0300010100 49ff00 00000000
Value: 0014000 6000a0300010100 0025ff 00000000
```

第一组显然是递增的字符串，可能是某种数据包的序列号（可以是写指令操作码或只是数据包号），后只是8个零的字符串。中间的字符串才是重点，和我猜的差不多，6字节的字符串，红色蓝色和绿色各两字节。

0025ff -> 00 25 ff (红色关闭 蓝色25 绿色满密度)
9ff00
ff0000
49ff00
0025ff 等等

No.	Time	Source	Destination	Protocol	Length	Info	
187	26.765234	localhost ()	TexasIns_caf0:36	ATT	12	Sent Read Request, Handle: 0x001e (Unknown: Unknown)	
188	26.782775	TexasIns_caf0:36	localhost ()	ATT	28	Rcvd Read Response, Handle: 0x001e (Unknown: Unknown)	
196	30.139151	localhost ()	TexasIns_caf0:36	ATT	30	Sent Write Request, Handle: 0x0012 (Unknown: Unknown)	
197	30.179944	TexasIns_caf0:36	localhost ()	ATT	10	Rcvd Write Response, Handle: 0x0012 (Unknown: Unknown)	
198	30.185487	localhost ()	TexasIns_caf0:36	ATT	12	Sent Read Request, Handle: 0x001e (Unknown: Unknown)	



通过这种方式，如果你想要产生蓝色，可以在8个0和UUID之间加上00ff00，如“00140006000a030001010000ff0000000000”，把这个通过BLE发给灯泡，地址已经从暴露的GATT中知道。理论上很简单，实际操作中，可能是反转的字符串，比如不是RGB而是BGR（另一种常见的颜色表示方式），其中0表示全亮度，255表示关闭。

现在我将使用Kali linux，如果你的电脑中有一个集成的蓝牙，或者你可以使用支持BLE (LE v4) 的USB蓝牙适配器，则可以控制灯泡。我很确定我的Macbook Air有一个。我设置了虚拟机，通过虚拟机的USB端口使用主机 (Macbook Air) 的内置蓝牙硬件，在这一点上，我不知道Kali是否支持驱动。赌一把，我竟然有一个支持蓝牙的树莓派，但是还是想先试一下前者。

在linux虚拟机上，我打开终端检查设备是否运行：

```
root@kali:~# hciconfig
hci0:  Type: Primary  Bus: USB
BD Address: E0:AC:CB:81:CE:37  ACL MTU: 1021:8  SCO MTU: 64:1
UP RUNNING
RX bytes:1859 acl:2 sco:0 events:106 errors:0
TX bytes:3059 acl:3 sco:0 commands:94 errors:0
```

开心！设备被检测到了！现在就要安装所需要的包了。

```
root@kali:~# apt-get install bluez bluez-hcidump, bluez-tools
```

运行下面的命令扫描BLE设备：

```
root@kali:~# hcitool lescan
LE Scan ...
88:C2:55:CA:F0:36 (unknown)
88:C2:55:CA:F0:36 Cnligh
```

注意：设备Cnligh是我们的灯泡，另一个设备可能是我的智能健身追踪器。

如果没找到，请检查灯泡是否打开，且不要将灯泡连接你的手机，因为它是P2P通信。

接下来我们连接设备：

```
root@kali:~# gatttool -I
[88:C2:55:CA:F0:36][LE]> connect 88:C2:55:CA:F0:36
Attempting to connect to 88:C2:55:CA:F0:36
Connection successful
[88:C2:55:CA:F0:36][LE]>
```

在这里，我使用gatttool Bluez程序通过地址连接到灯泡。

注意：如果出现任何错误，在/etc/bluetooth/main.conf中添加以下内容：

```
EnableLE = true           // Enable Low Energy support. Default is false.
AttributeServer = true    // Enable the GATT attribute server. Default is false.
```

然后重启：

```
root@kali:~# etc/init.d/bluetooth restart
```

再次尝试，应该就能成功了。

一旦你连上了灯泡，你就能发现更多：

[88:C2:55:CA:F0:36][LE]> help	
help	Show this help
exit	Exit interactive mode
quit	Exit interactive mode
connect [address [address type]]	Connect to a remote device
disconnect	Disconnect from a remote device
primary [UUID]	Primary Service Discovery
included [start hnd [end hnd]]	Find Included Services

characteristics [start hnd [end hnd [UUID]]]	Characteristics Discovery
char-desc [start hnd] [end hnd]	Characteristics Descriptor Discover
char-read-hnd <handle>	Characteristics Value/Descriptor Re
char-read-uuid <UUID> [start hnd] [end hnd]	Characteristics Value/Descriptor Re
char-write-req <handle> <new value>	Characteristic Value Write (Write F
char-write-cmd <handle> <new value>	Characteristic Value Write (No resp
sec-level [low medium high]	Set security level. Default: low
mtu <value>	Exchange MTU for GATT/ATT

```
[88:C2:55:CA:F0:36][LE]> primary
attr handle: 0x0001, end grp handle: 0x000b uuid: 00001800-0000-1000-8000-00805f91
attr handle: 0x000c, end grp handle: 0x000f uuid: 00001801-0000-1000-8000-00805f91
attr handle: 0x0010, end grp handle: 0xffff uuid: 0000f371-0000-1000-8000-00805f91
```

```
[88:C2:55:CA:F0:36][LE]> characteristics
handle: 0x0002, char properties: 0x02, char value handle: 0x0003, uuid: 00002a00-0000-1000-8000-00805f91
handle: 0x0004, char properties: 0x02, char value handle: 0x0005, uuid: 00002a01-0000-1000-8000-00805f91
handle: 0x0006, char properties: 0x0a, char value handle: 0x0007, uuid: 00002a02-0000-1000-8000-00805f91
handle: 0x0008, char properties: 0x08, char value handle: 0x0009, uuid: 00002a03-0000-1000-8000-00805f91
handle: 0x000a, char properties: 0x02, char value handle: 0x000b, uuid: 00002a04-0000-1000-8000-00805f91
handle: 0x000d, char properties: 0x20, char value handle: 0x000e, uuid: 00002a05-0000-1000-8000-00805f91
handle: 0x0011, char properties: 0x0a, char value handle: 0x0012, uuid: 0000ffff1-0000-1000-8000-00805f91
handle: 0x0014, char properties: 0x0a, char value handle: 0x0015, uuid: 0000ffff2-0000-1000-8000-00805f91
handle: 0x0017, char properties: 0x0a, char value handle: 0x0018, uuid: 0000ffff3-0000-1000-8000-00805f91
handle: 0x001a, char properties: 0x0a, char value handle: 0x001b, uuid: 0000ffff4-0000-1000-8000-00805f91
handle: 0x001d, char properties: 0x0a, char value handle: 0x001e, uuid: 0000ffff5-0000-1000-8000-00805f91
handle: 0x0020, char properties: 0x0a, char value handle: 0x0021, uuid: 0000ffff6-0000-1000-8000-00805f91
handle: 0x0023, char properties: 0x0a, char value handle: 0x0024, uuid: 0000ffff7-0000-1000-8000-00805f91
handle: 0x0026, char properties: 0x0a, char value handle: 0x0027, uuid: 0000ffff8-0000-1000-8000-00805f91
handle: 0x0029, char properties: 0x10, char value handle: 0x002a, uuid: 0000ffff9-0000-1000-8000-00805f91
[88:C2:55:CA:F0:36][LE]>
```

现在我们尝试通过发送下方字符串来改变灯泡颜色:

```
[88:C2:55:CA:F0:36][LE]> char-write-cmd 0x0012 00140006000a0300010100ff0000000000000000
```

然后果然变成红色了！！！ 视频在下面。

再试试蓝色：

```
[88:C2:55:CA:F0:36][LE]> char-write-cmd 0x0012 00140006000a030001010000ff0000000000
```

再试试绿色：

```
[88:C2:55:CA:F0:36][LE]> char-write-cmd 0x0012 00140006000a03000101000000ff00000000
```

一切都在按预期发生。

演示

为了演示工作，我准备了一个小的bash脚本，将灯泡的颜色循环到红色 -> 绿色 -> 蓝色 -> 白色 -> 红色

```
#!/bin/bash

echo "Controlling SYSKA Smart light bulb"
sleep 3
echo "Look Mah! No App

"

sleep 3

while true;
do
gatttool -i hci0 -b 88:C2:55:CA:F0:36 --char-write-req -a 0x0012 -n 00140006000a030001010000ff0000000000
sleep 3
echo "RED"
gatttool -i hci0 -b 88:C2:55:CA:F0:36 --char-write-req -a 0x0012 -n 00140006000a030001010000ff0000000000
sleep 3
echo "GREEN"
gatttool -i hci0 -b 88:C2:55:CA:F0:36 --char-write-req -a 0x0012 -n 00140006000a030001010000ff0000000000
sleep 3
echo "BLUE"
gatttool -i hci0 -b 88:C2:55:CA:F0:36 --char-write-req -a 0x0012 -n 00140006000a030001010000ff0000000000
```

```

sleep 3
echo "WHITE"
gatttool -i hci0 -b 88:C2:55:CA:F0:36 --char-write-req -a 0x0012 -n 00100006000a0:
sleep 3
echo "OFF"
gatttool -i hci0 -b 88:C2:55:CA:F0:36 --char-write-req -a 0x0012 -n 00100006000a0:
done
exit 0

```

<

[GitHub](#)上也有。

您可以使用颜色选择器（如[HTML颜色选择器](#)）中的RGB值来模拟任何颜色。

视频

https://www.youtube.com/watch?v=g1_8cY-fIM&feature=youtu.be

思考

虽然这是一个有趣的逆向IoT设备的功能的项目，但看到数据包不是加密的，我仍然很伤心，很多公司专注于缩短他们的物联网产品的上市时间，但在这个过程中，他们并没有采取最大的措施来确保设备的安全，只是一个蓝牙智能电灯泡，不会搞出大新闻，黑客最多也只能改变你家里的灯泡颜色，但是如果智能锁也这样，那么麻烦就大了，你的车库和家门就会受到很大威胁。或者一个基于tcp的设备，可以在DDoS攻击时充当僵尸网络的一员，令人失望的是生产商只是简单地参照了示例代码，甚至不改变蓝牙协议本身提供的128位UUID灵活性。

下一步

我现在有了API，正在计划一个小型项目，从Yahoo Weather或Weather Underground等在线服务中获得我的位置的天气数据，并使我的灯泡天气变化实时反应，例如雨天呼吸蓝色，阳光充足就用白色，温暖的天气用橙色。我可以使用Python Bluez API (pyBlue或其他)。请保持关注，另外，感谢阅读：)

*参考来源：[github](#)，Covfefe编译，转载请注明来自 FreeBuf.COM

上一篇：[“金蝉脱壳”病毒分析报告](#)

下一篇：[机器人是如何被入侵并被用于监视和破坏的](#)

已有 5 条评论

[freebug_n](#) (4级) You hava no body! 2017-08-31

[1楼 回](#)

666

[亮了](#)

[FelixxX](#) 2017-08-31

[2楼 回](#)

我想请教一下大神，这需要哪些方面的知识与技能，看了这篇文章对于我这样的小白来说，只能用惊叹来形容

[亮了](#)

[Covfefe](#) (6级) So incovfefeable! 2017-09-01

[

@ FelixxX 研究研究无线电吧

[亮了](#)

[knowsec](#) 2017-09-01

[3楼 回](#)

这种BLE的文章并没任何高技术含量，全文无亮点是痛点

[亮了](#)

[Covfefe](#) (6级) So incovfefeable! 2017-09-01

[

@ knowsec 看来这种文章得加新手科普了，以后选文章会注意 (QAQ)

[亮了](#)

必须 您当前尚未登录。[登陆?](#) [注册](#)

昵称

请输入昵称

邮箱

必须 (保密)

请输入邮箱地址

表情 插图

[取消](#)



有人回复时邮件通知我



[Covfefe](#)

So incovfefeable!

42
文章数

12
评论数

最近文章

[如何绕过安卓的网络安全配置功能](#)

2018.03.31

[如何搭配USRP在安卓设备上搭建GNU Radio](#)

2018.03.22

[勒索未去挖矿又来，解读网络淘金热的黑暗面](#)

2018.02.25

[浏览更多](#)

相关阅读

[2014上半年国内安卓银行应用隐私泄...](#)

[技术分析：“厄运cookie”漏洞\(CV...](#)

[看我如何从漏洞公告入手黑掉一台打印...](#)

[极客DIY：打造属于自己的无线移动渗...](#)

[在Linux上使用mojoqq来实现命令行...](#)

特别推荐



关注我们 分享每日精选文章

活动预告

<p>5月 北京 <u>DEF CON China极客大会</u></p> <p>未开始</p> <p>5月 <u>【火热报名中】鸡肋漏洞的深思： CORS、XSS、CSRF的组合</u></p> <p>未开始</p>	<p>5月 北京 <u>Freetalk2018北京站</u></p> <p>未开始</p> <p>4月 <u>【已结束】构建和运行SOC的最佳 实践</u></p> <p>已结束</p>
---	--



Copyright © 2018 WWW.FREEBUF.COM All Rights Reserved [沪ICP备13033796号](#)

 阿里云 提供计算与安全服务