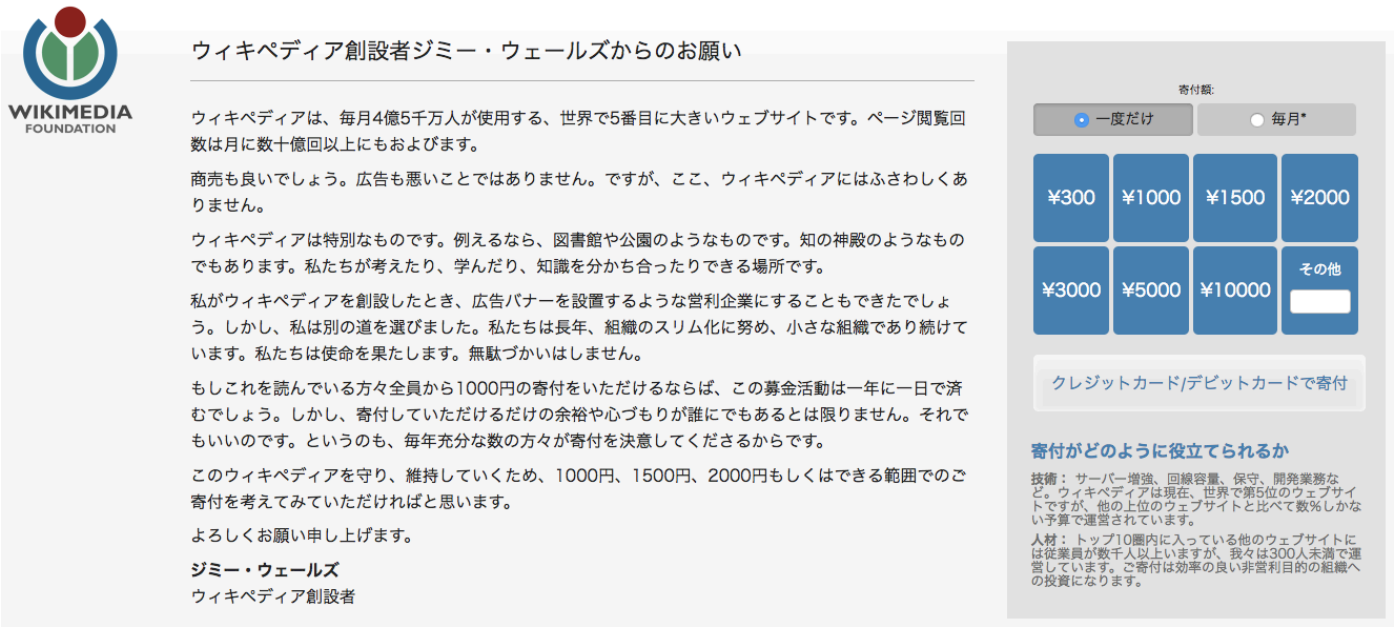
黒林檎のお部屋

-IoT のハッキング編-

[このドキュメントについてについて]
IoT 系の勉強会開催するから事前に学習したいという方向けに作成した資料なので、どう使って頂いても自由です。簡単に言えば、フリー公開ですがこの資料で得た知識で逮捕や炎上しても知りません。勉強会などで資料作るのだるいしコレ使いたいけど後で文句言われたら怖いなーという感じでしたら、twitter の@r00tapple に声かけてくれれば大丈夫です。

このドキュメントはお金を取りませんが、業界のノリに便乗してウィキペディアの寄付先の URL アドレス貼っときますので気が向いたら満足度分程度寄付してください。



(図)Wikipedia 寄付先リンク

wiki 寄付リンク:
https://donate.wikimedia.org/w/index.php?title=Special:FundraiserLandingPage&country=JP&uselang=ja&utm_medium=sidebar&utm_source=donate&utm_campaign=C13_ja.wikipedia.org

[+] ドキュメント作成者の自己紹介

ハンドルネーム　：黒林檎

Twitter　：r00tapple

Facebook　：村島正浩

Mail　：packr@packr.org

ROOTAPPLE

Bluetooth のハッキング

■はじめに

最近では、色々なところに Bluetooth が用いられています。
例えば、車のカーナビや自動販売機のポイントシステムなどが挙げられます。
非 IT 技術者でも、"Bluetooth"という単語を口にする事を珍しいことではありません。
では、Bluetooth とは何でしょうか?
Wikipedia によると、デジタル機器用の近距離無線通信規格の 1 つであるとあります。
筆者の場合、無線通信と聞いてすぐに思いつくのは無線 LAN です。
無線 LAN の解析はある程度まとまったドキュメントが多数あります。
Bluetooth のハッキングに関するまとまったドキュメントはどうでしょう?
筆者は見かけた事がありません。
良い機会なの日本語の Bluetooht のハッキングについて簡単ですがまとめてみましょう。

■対象機器

今回 Bluetooth のハッキング対象となるのは以下の 2 つの機器です。

電球(Bluetooth 4.0)

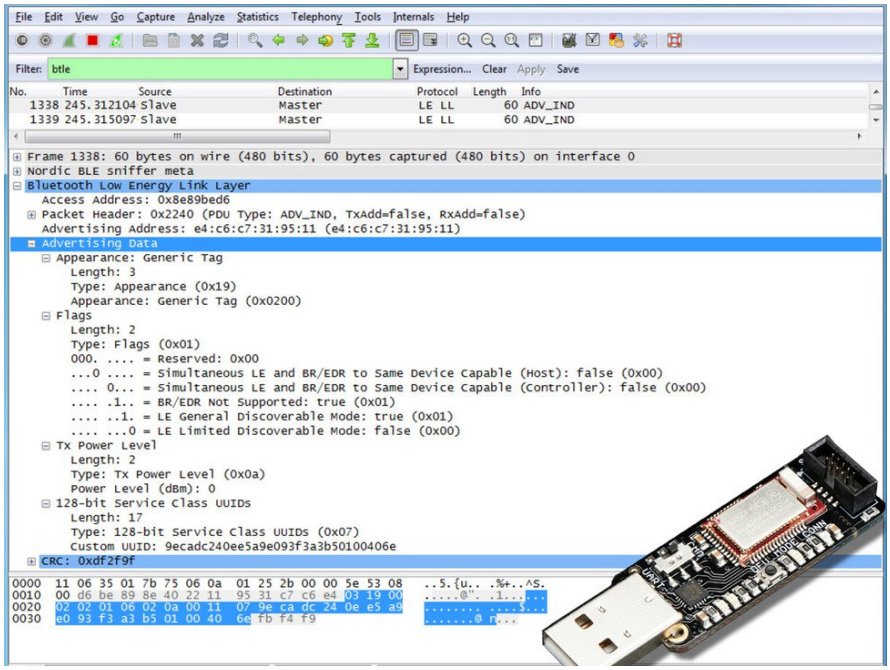
購入サイト:
https://www.amazon.co.jp/gp/product/B0142IKRDU/ref=oh_aui_detailpage_o04_s00?ie=UTF8&psc=1

時計(Bluetooth 4.0)

購入サイト:
https://www.amazon.co.jp/gp/product/B017HIWS7K/ref=oh_aui_detailpage_o04_s00?ie=UTF8&psc=1

■Bluetooth ハッキングの基礎知識

Bluetooth のハッキングと聞くと難しく感じます。
数年前筆者は Bluetooth のハッキングに憧れて、モバイルに潜む危機という本など読みましたが思うような Bluetooth のハッキングは出来ませんでした。
しかし、思っているより Bluetooth のハッキングとは簡単です。
Bluetooth でペアリングが必要ない機器があります。
例えば Bluetooth 電球などは多くの場合ペアリングキーは不要です。
ペアリングキーが不要であった場合、gatttool というユーザーインターフェースで簡単に接続することが出来ます。
Bluetooth のパケットはどの様に取得出来るでしょうか?
無線 LAN クラックの場合は、airodump というパケット収集ツールを使用します。
結論からいうと無差別な Bluetooth パケットを取得するには専用の機器が必要になります。
Bluefruit LE Sniffer - Bluetooth Low Energy (BLE 4.0) - nRF51822 - v1.0 という Bluetooth パケットスニッファが海外で販売されています。
このパケットスニッファを用いることで無差別な Bluetooth パケットを収集することが出来ます。

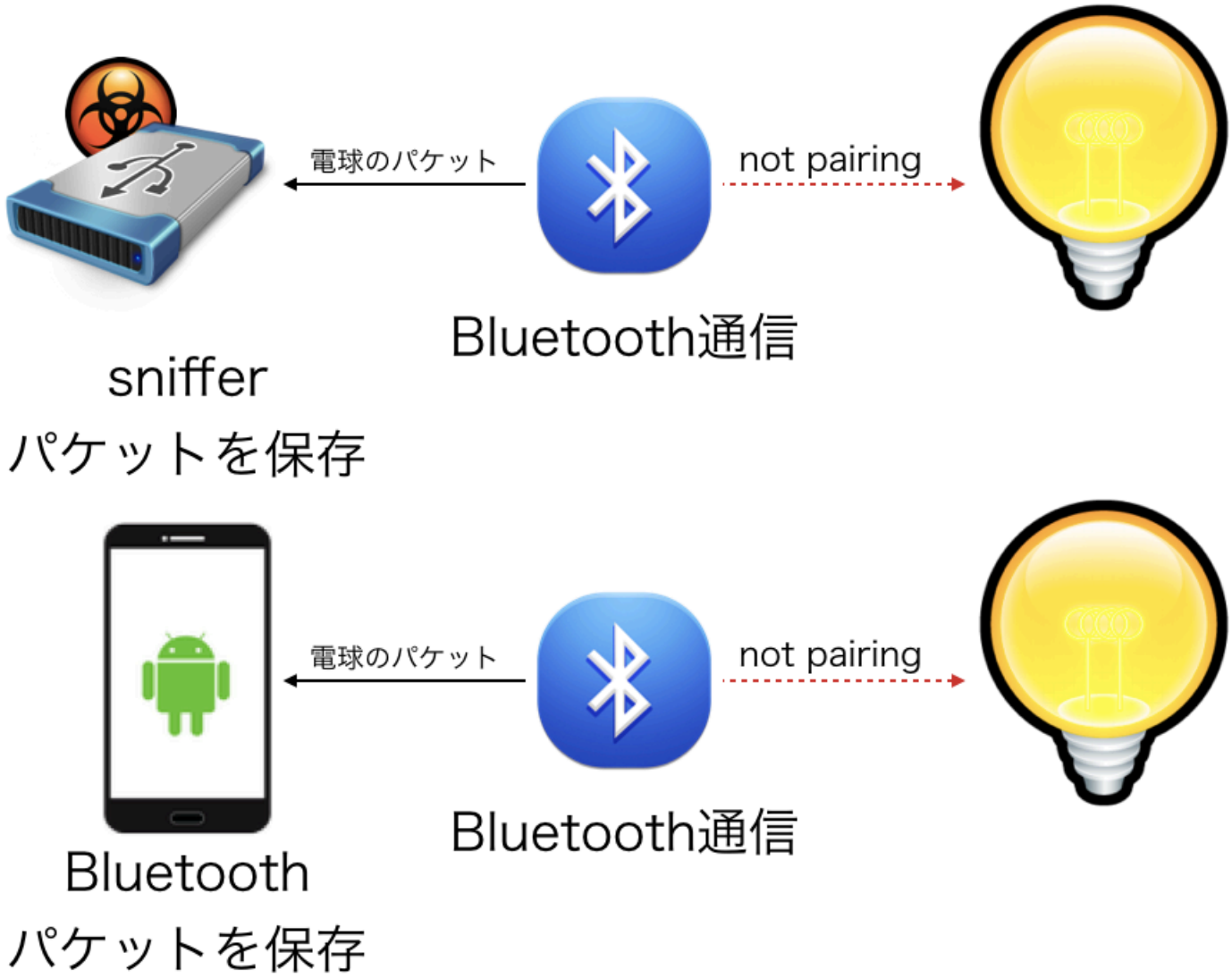


(図) Bluefruit LE Sniffer

"**無差別な Bluetooth パケット**"であれば確かに Bluefruit LE Sniffer などの専用機器が必要ですが、実際 Bluefruit LE Sniffer で取得出来るパケットは攻撃に使えるイメージがありません。
理由ですが、Android 端末と Bluetooth 機器がペアリングした状態であれば Android 端末側で送受信している Bluetooth パケットを HCI スヌープログという機能で取得する事が出来るからです。
HCI スヌープログで Bluetooth パケットを保存する時は以下の様にペアリングしている状況でなければなりません。

(図)Android 端末で Bluetooth パケットを保存する時のイメージ図

例えば、下記の様にペアリングをしていない状態であれば Bluefruit LE Sniffer ではパケットを取得できますが、HCI スヌープログという機能では正常にパケットは取得できません。



(図)スニッファと Android 端末でのパケット取得の差

HCI スヌープログという言葉が数回出てきましたが、HCI スヌープログとは何でしょうか?
このオプションを使用する事で、BlueZ の hcidump 相当のログが出力されます。
ログの保存場所は、機器により異なる場合もありますが以下の様に adb shell で HCI スヌープログを使用する機器に接続し設定ファイルを確認する事で簡単に調べる事が出来ます。
太字にしている所が重要なポイントです。

```
##########################################################
shell@ZTE_Blade_A476:/ $ cat /etc/bluetooth/bt_stack.conf
# Enable BtSnoop logging function
# valid value：true, false
BtSnoopLogOutput=false

# BtSnoop log output file
BtSnoopFileName=/sdcard/btsnoop_hci.log
```

# Preserve existing BtSnoop log before overwriting
BtSnoopSaveLog=false

# Enable trace level reconfiguration function
# Must be present before any TRC_ trace level settings
TraceConf=true

# Trace level configuration
# BT_TRACE_LEVEL_NONE 0 （No trace messages to be generated）
# BT_TRACE_LEVEL_ERROR 1 （Error condition trace messages）
# BT_TRACE_LEVEL_WARNING 2 （Warning condition trace messages）
# BT_TRACE_LEVEL_API 3 （API traces）
# BT_TRACE_LEVEL_EVENT 4 （Debug messages for events）
# BT_TRACE_LEVEL_DEBUG 5 （Full debug messages）
# BT_TRACE_LEVEL_VERBOSE 6 （Verbose messages）- Currently supported for TRC_BTAPP only.
TRC_BTM=2
TRC_HCI=2
TRC_L2CAP=2
TRC_RFCOMM=2
TRC_OBEX=2
TRC_AVCT=2
TRC_AVDT=2
TRC_AVRC=2
TRC_AVDT_SCB=2
TRC_AVDT_CCB=2
TRC_A2D=2
TRC_SDP=2
TRC_GATT=2
TRC_SMP=2
TRC_BTAPP=2
TRC_BTIF=2
##

では、HCI スヌープログはどの様に使用すれば良いのでしょうか?
使い方は簡単で、Android の開発者向けオプション(開発者向けオプションが表示されていない方は、設定->端末情報->[ビルド番号]を何かアラートが出るまでクリックしてください。)

そうすると下記の様に HCI スヌープログを利用できる様になります。
この機能は Android4.4 からしいのでそれ以前のバージョンを利用している方は利用できない場合があります。

(図)開発者向けオプションに HCI スヌープログを使う項目がある。

では、この HCI スヌープログで実際に先ほど紹介した 2 つの IoT 機器のパケットを取得しました。
以下の Google Drive に公開されているのでお好きにダウンロードして解析してください。

##
パケット公開先:
https://drive.google.com/drive/folders/0B2ENdYkFyZVESzlNMm5PWDJKaDQ?usp=sharing
パケット公開先(短縮 URL 版):
goo.gl/om2PMK
##

それではこれから実際に各機器をハッキングしていきましょう。

電球のハッキング

■電球のハッキング

Bluetooth の初めてのハッキングは電球をお勧めしています。単純でやりやすいという理由です。実際に解析した時のコツなどを書いていきます。

電球の概要

- 照明が付いていなくても電源接続時は常時 Bluetooth パケットを受け付ける
- Android アプリ側からペアリングを求められない(ペアリングキーが無い)

電球のハッキングを大まかにすると、この 2 つでしょう。
この 2 点から Bluetooth のパケットさえ再現出来れば簡単に電球を自由に ON-OFF する事が出来ると考えられます。
そのためには、Bluetooth パケットを取得して解析しどの様な値を投げているか確認する必要性があります。
そもそも Bluetooth のパケットは特性上、様々なデバイスでの通信に使用される為、機器の種類ごとに策定されたプロトコルがあり、これをプロファイルと呼び標準化しており、用途により異なるプロファイラが使用される。

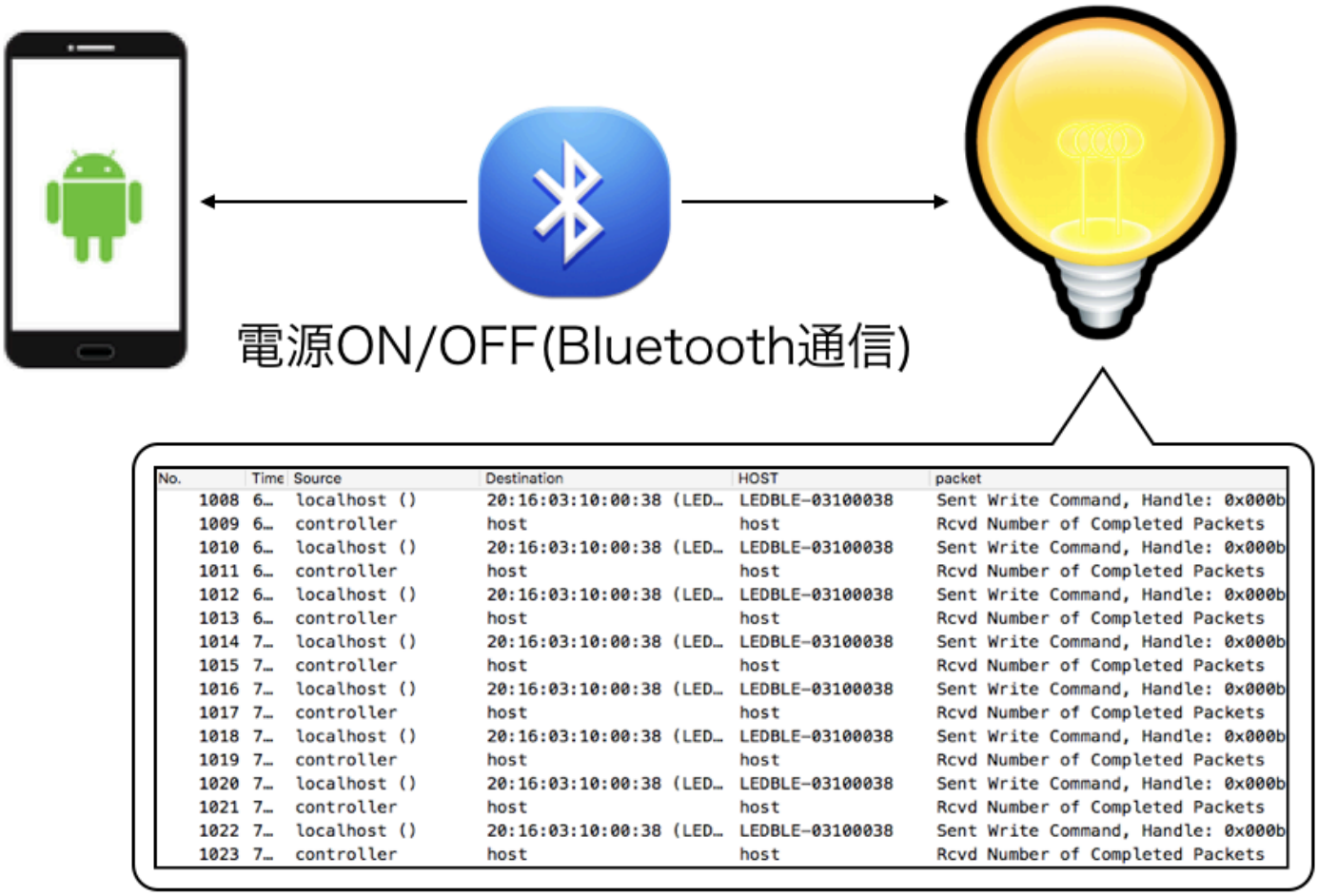
##
GAP (Generic Access Profile)
機器の接続/認証/暗号化を行うためのプロファイル。

HID (Human Interface Device Profile)
マウスやキーボードなどの入力機器を無線化するためのプロファイル。

PBAP (Phone Book Access Profile)
電話帳のデータを転送するためのプロファイル
##

Generic Attributes (GATT) は、接続された Bluetooth LE デバイスに提示された階層データ構造を定義し、GATT はこれらのサービスをグループ化して、デバイスの一部の動作をカプセル化し、GATT 機能に基づいたユースケース、ロール、一般動作を記述します。このフレームワークは、サービスとその特性のプロシージャと形式を定義します。これには特性の検出、**読み取り、書き込み、通知、提示のほか、特性のブロードキャスト設定が含まれます。**

Bluetooth のパケット構造についてはちゃんと勉強したほうが良さそうですが、とりあえずココらへんの IoT のハッキングではそこまでの認識は必要ありません。
一番早いのは慣れる事です、下記図の様にしてください。



(図)HCI スヌープログを ON にした状態で電源の ON/OFF を繰り返す。

HCI スヌープログを ON にした状態で、Android 端末にインストールされている電球操作用のアプリケーションで電球を ON/OFF を数回繰り返します。
保存されたパケットは以下の様な状態になります。

No.	Time	Source	Destination	HOST	packet
1008	6…	localhost ()	20:16:03:10:00:38 (LED…	LEDBLE-03100038	Sent Write Command, Handle: 0x000b
1009	6…	controller	host	host	Rcvd Number of Completed Packets
1010	6…	localhost ()	20:16:03:10:00:38 (LED…	LEDBLE-03100038	Sent Write Command, Handle: 0x000b
1011	6…	controller	host	host	Rcvd Number of Completed Packets
1012	6…	localhost ()	20:16:03:10:00:38 (LED…	LEDBLE-03100038	Sent Write Command, Handle: 0x000b
1013	6…	controller	host	host	Rcvd Number of Completed Packets
1014	7…	localhost ()	20:16:03:10:00:38 (LED…	LEDBLE-03100038	Sent Write Command, Handle: 0x000b
1015	7…	controller	host	host	Rcvd Number of Completed Packets
1016	7…	localhost ()	20:16:03:10:00:38 (LED…	LEDBLE-03100038	Sent Write Command, Handle: 0x000b
1017	7…	controller	host	host	Rcvd Number of Completed Packets
1018	7…	localhost ()	20:16:03:10:00:38 (LED…	LEDBLE-03100038	Sent Write Command, Handle: 0x000b
1019	7…	controller	host	host	Rcvd Number of Completed Packets
1020	7…	localhost ()	20:16:03:10:00:38 (LED…	LEDBLE-03100038	Sent Write Command, Handle: 0x000b
1021	7…	controller	host	host	Rcvd Number of Completed Packets
1022	7…	localhost ()	20:16:03:10:00:38 (LED…	LEDBLE-03100038	Sent Write Command, Handle: 0x000b
1023	7…	controller	host	host	Rcvd Number of Completed Packets

(図)電源 ON/OFF を繰り返してる際の Bluetooth パケット

初めて見るパケットなので、意味不明な通信なども多数ありますが自分で識別できるであろう箇所を作成する事で簡単に該当箇所を見つける事が可能です。

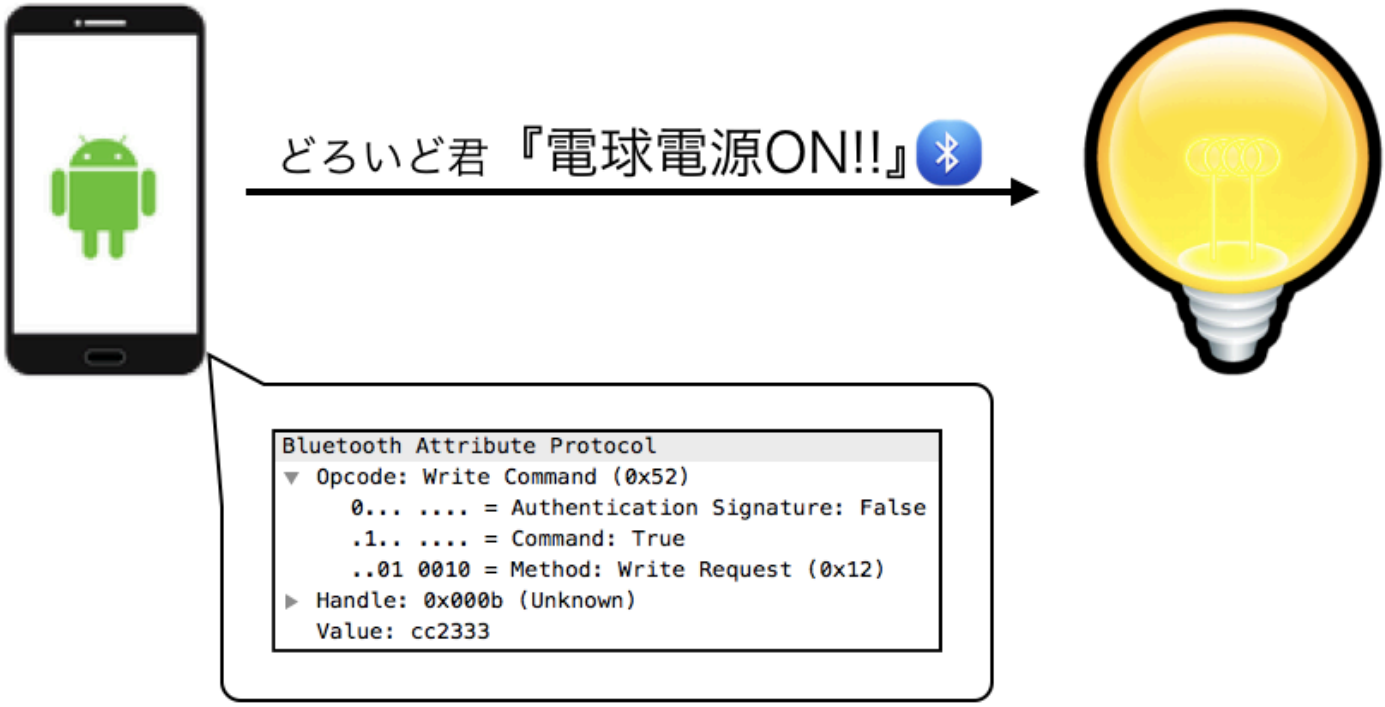
No.	Time	Source	Destination	HOST	packet
133	7…	controller	host	host	Rcvd LE Meta (LE Advertising Report)
134	7…	host	controller	controller	Sent LE Create Connection
135	7…	controller	host	host	Rcvd Command Status (LE Create Connection)
136	7…	controller	host	host	Rcvd LE Meta (LE Connection Complete)
137	7…	host	controller	controller	Sent LE Read Remote Used Features
138	7…	localhost ()	20:16:03:10:00:38 (LED…	LEDBLE-03100038	Sent Read By Group Type Request, GATT Primary Service Declaration, Handles: 0x0001..0xffff
139	7…	controller	host	host	Rcvd Command Status (LE Read Remote Used Features)
140	7…	host	controller	controller	Sent LE Add Device To White List
141	7…	controller	host	host	Rcvd Command Complete (LE Add Device To White List)
142	7…	20:16:03:10:00:3…	localhost ()		Rcvd Connection Parameter Update Request
143	7…	localhost ()	20:16:03:10:00:38 (LED…	LEDBLE-03100038	Sent Connection Parameter Update Response (Accepted)
144	7…	controller	host	host	Rcvd LE Meta (LE Read Remote Used Features Complete)
145	7…	controller	host	host	Rcvd Number of Completed Packets
146	7…	20:16:03:10:00:3…	localhost ()		Rcvd Read By Group Type Response, Attribute List Length: 3
147	7…	localhost ()	20:16:03:10:00:38 (LED…	LEDBLE-03100038	Sent Read By Group Type Request, GATT Primary Service Declaration, Handles: 0x000c..0xffff
148	7…	controller	host	host	Rcvd Number of Completed Packets

上記の様に少し読むのに時間が掛かりそうな部分などを一先ず読まずに結果を出せます。
電球の ON/OFF している際のパケットは以下の様になっています。

```
Bluetooth Attribute Protocol
▼ Opcode: Write Command (0x52)
      0... .... = Authentication Signature: False
      .1.. .... = Command: True
      ..01 0010 = Method: Write Request (0x12)
▶ Handle: 0x000b (Unknown)
  Value: cc2333
```

(図)電球 ON 時に送信するパケット

おそらく、Android 端末は電球操作のアプリで送信する value 値など含んだ Bluetooth パケットを以下の様に送信しているのでしょう。



(図)電源 ON 時のパケット送信予想図

この Bluetooth パケット再現出来れば Android 端末でなくパソコンから命令を送信できるはずです。
再現手順は簡単です、以下の様に再現してください。
まず、hciconfig コマンドで Bluetooth を発見できるディバイスを表示します。

```
root@root:~# hciconfig
hci1:   Type: BR/EDR  Bus: USB
        BD Address: 5C:F3:70:79:53:6F  ACL MTU: 1021:8  SCO MTU: 64:1
        UP RUNNING
        RX bytes:948 acl:0 sco:0 events:47 errors:0
        TX bytes:1495 acl:0 sco:0 commands:47 errors:0

hci0:   Type: BR/EDR  Bus: USB
        BD Address: 00:50:56:F3:5D:F8  ACL MTU: 8192:128  SCO MTU: 64:128
        UP RUNNING
        RX bytes:533 acl:0 sco:0 events:27 errors:0
        TX bytes:609 acl:0 sco:0 commands:27 errors:0
```

(図)hciconfig 実行結果

"""
hciconfig の概要
$man hciconfig

hciconfig is used to configure Bluetooth devices. hciX is the name of a Bluetooth device installed in the system. If hciX is not given, hci-config prints name and basic information about all the Bluetooth devices installed in the system. If hciX is given but no command is given, it prints basic information on device hciX only. Basic information is interface type, BD address, ACL MTU, SCO MTU, flags (up, init, running, raw, page scan enabled, inquiry scan enabled, inquiry, authentication enabled, encryption enabled).

"""

次に、hcitool scan を使って周辺の Bluetooth の Mac アドレスなどを調べます。

```
root@root:~# hcitool lescan
LE Scan ...
20:16:03:10:00:38 (unknown)
20:16:03:10:00:38 LEDBLE-03100038
```

(図)hcitool lescan

hcitool は lescan 以外に scan を指定できます。
hcitool scan
　　-scan　Scan for remote devices(通常の Bluetooth ディバイス)
hcitool lescan
　　- lescan　Start LE scan(Bluetooth LE の場合)

Bluetooth LE とは?
"""
Bluetooth LE(Bluetooth Low Energy)とは、無線 PAN 技術である Bluetooth の仕様における、バージョン 4.0 の呼称である。
バージョン 3.0 までの Bluetooth と比較して、省電力かつ省コストで通信や実装を行うことを意図して設計されている。3.0 までに策定されている仕様からは独立しており、後方互換性は持たないが、3.0 までの Bluetooth 仕様との同居は可能である。もとの仕様は Wibree という名称で 2006 年に Nokia によって開発されたものであり[2]、これが 2010 年に Bluetooth 4.0 として統合された。
参考: https://ja.wikipedia.org/wiki/Bluetooth_Low_Energy
"""
要するに、hcitool scan は Bluetooth 3.0 などに対して使用して Bluetooth 4.0 からは Bluetooth lescan になるのでしょう。

対象の Bluetooth の Mac アドレスが判明した事で、パケットの送信先がわかりました。
そんな面倒な手順はゴメンだ、という方もいるでしょう。
勿論、Wireshark でパケットを読み込めるので以下の様に送信元:送信先を調べる事が出来ます。

```
Bluetooth
    [Source: 00:00:00_00:00:00 (00:00:00:00:00:00)]
    [Destination: 20:16:03:10:00:38 (20:16:03:10:00:38)]
```

(図)Wireshark で送信元-送信先を調べた

あとは、gatttool を使用すれば対話的なユーザーインタフェースで Bluetooth 機器などと接続して任意のパケットを送信する事が出来ます。

"""
$man gatttool
gatttool is tool that can be used to manipulate these attributes with a Bluetooth Low Energy device.
"""

gatttol で電球の電源を操作するのは以下です。
gatttool -i hci0 -b 20:16:03:10:00:38 -I
というコマンドは..
gatttool -i 使用するインターフェース -b 対象の MAC アドレス インタラクティブモード
を命令しています。

```
root@root:~# gatttool -i hci0 -b 20:16:03:10:00:38 -I
[20:16:03:10:00:38][LE]> connect
Attempting to connect to 20:16:03:10:00:38
Connection successful
[20:16:03:10:00:38][LE]> primary
attr handle: 0x0001, end grp handle: 0x0007 uuid: 00001800-0000-1000-8000-00805f9b34fb
attr handle: 0x0008, end grp handle: 0x0008 uuid: 0000fff0-0000-1000-8000-00805f9b34fb
attr handle: 0x0009, end grp handle: 0x000b uuid: 0000ffe5-0000-1000-8000-00805f9b34fb
attr handle: 0x000c, end grp handle: 0xffff uuid: 0000ffe0-0000-1000-8000-00805f9b34fb
[20:16:03:10:00:38][LE]> char-write-cmd 0x000b cc2433
[20:16:03:10:00:38][LE]> char-write-cmd 0x000b cc2433
[20:16:03:10:00:38][LE]> char-write-cmd 0x000b cc2333
[20:16:03:10:00:38][LE]> char-write-cmd 0x000b cc2433
[20:16:03:10:00:38][LE]> char-write-cmd 0x000b cc2333
[20:16:03:10:00:38][LE]>
```

(図)gatttool を使用して電球操作

無事に電球に接続し、primary という文字を最初に入力し実行しました。
この primary というのはデバイスから取得出来るサービスの一覧です。
uuid の最下位 16 ビットは GATT が指定している値で以下の様になります。

##
attr handle = 0x0001, end grp handle = 0x0007 uuid: 0000**1800**-0000-1000-8000-00805f9b34fb
attr handle = 0x0011, end grp handle = 0xffff uuid: 0000**180f**-0000-1000-8000-00805f9b34fb

1800=Generic Access
180F=Battery Service
詳細参考リンク:
https://www.bluetooth.com/ja-jp/specifications/gatt/services
##

それでは、この primary から情報を取得する場合は gatttol で以下の様に実行します。

1800 の Generic Access を調べます。
Generic Access の handle は 0x0001 から 0x0007 であるため gatttool のインタラクティブモードで下記の様にして指定実行する事で

##

[LE]> char-desc 0x0001 0x0007
handle: 0x0001, uuid: 2800
handle: 0x0002, uuid: 2803
handle: 0x0003, uuid: 2a00
handle: 0x0004, uuid: 2803
handle: 0x0005, uuid: 2a01
詳細参考:
https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.service.generic_access.xml&u=org.bluetooth.service.generic_access.xml
##

primary は、Generic Access には、Bluetooht 機器の Device Name の Assigned Number: 0x2A00 とあります。
先ほど primary の uuid に 0x2A00(2a00)がありました、またその handle は 0x00003 でした。

##
[LE]> char-read-hnd 0x0003
Characteristic value/descriptor: B0 10 65 69 70 68 32 44 D1 FF 31 81
$ echo " B0 10 65 69 70 68 32 44 D1 FF 31 81"| tr ¥ = | nkf -WwmQ
LEDBLE-03100038
##

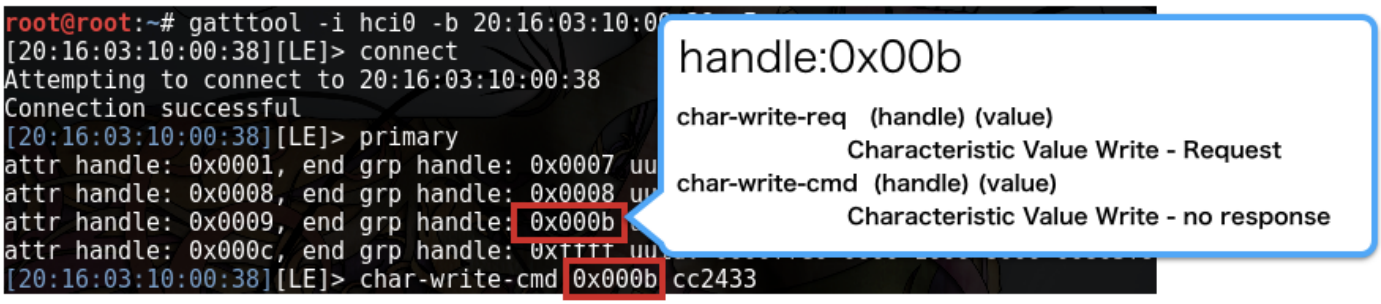
primary ではバッテリー情報などの確認も出来る様です、もう少し学んでいけば面白い事が出来そうな内容です。

それでは、最後に電球の電気操作する命令を送信します。
char-write-cmd という命令で送信しますが、0x00b の uuid は fff5 でした、忘れた方は"(図)gatttool を使用した電球操作"という一ページ前の図を確認してください。

GATT 的には、受信-送信は下記の uuid の様です。

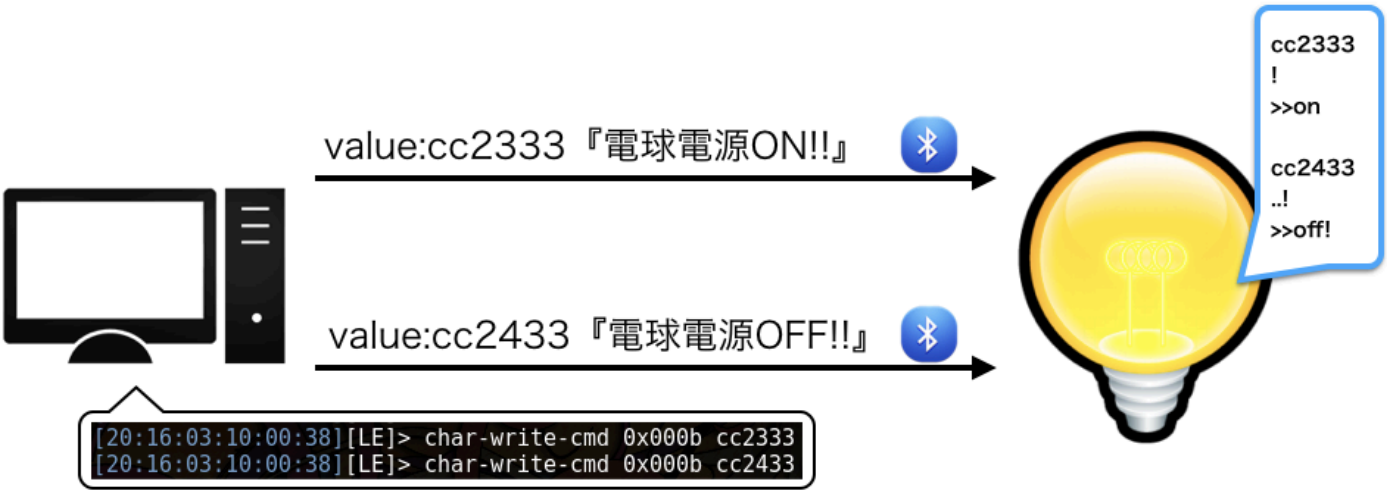
- Receive Data Service: 0xFFE0
- Send Data Service: 0xFFE5

よって uuid が 0xff5 の handle は 0x000b でした。
そのため、charwrite では引数に handle(今回は Send Data Service の handle)と value(クライアントが自由に読み書き出来る値で送信するデータ)を指定します。
その際の実行画面などが下記になります。



(図)char-write-cmd で命令実行

ついでに、value には最大 512bytes まで格納でき、整数、文字列、小数点と様々な型のデータが使えるらしいです。
char-write-cmd を上手に使用することにより、目的であるパソコン上からの任意の操作に成功しました。



(図)パソコンにより Bluetooth 機器の操作に成功

[コメント]
筆者の環境は Kali Linux なのですが、OS のアップデートなどで Bluez を新しくしなければ正常に接続出来なかったのは Bluetooth LE 非対応の PC を無理やり対応させて実行させたからか気になります。
もし、上手く gatttool で接続出来ない場合は OS をアップデートして

$sudo apt-get -y update && sudo apt-get -y upgrade && sudo apt-get -y dist-upgrade
$sudo rm /usr/local/bin/*
$sudo apt-get install bluetooth bluez blueman
$sudo apt-get install bluez-hcidump

で出来る様になると思います。

ROOTAPPLE

時計のハッキング

■時計のハッキング

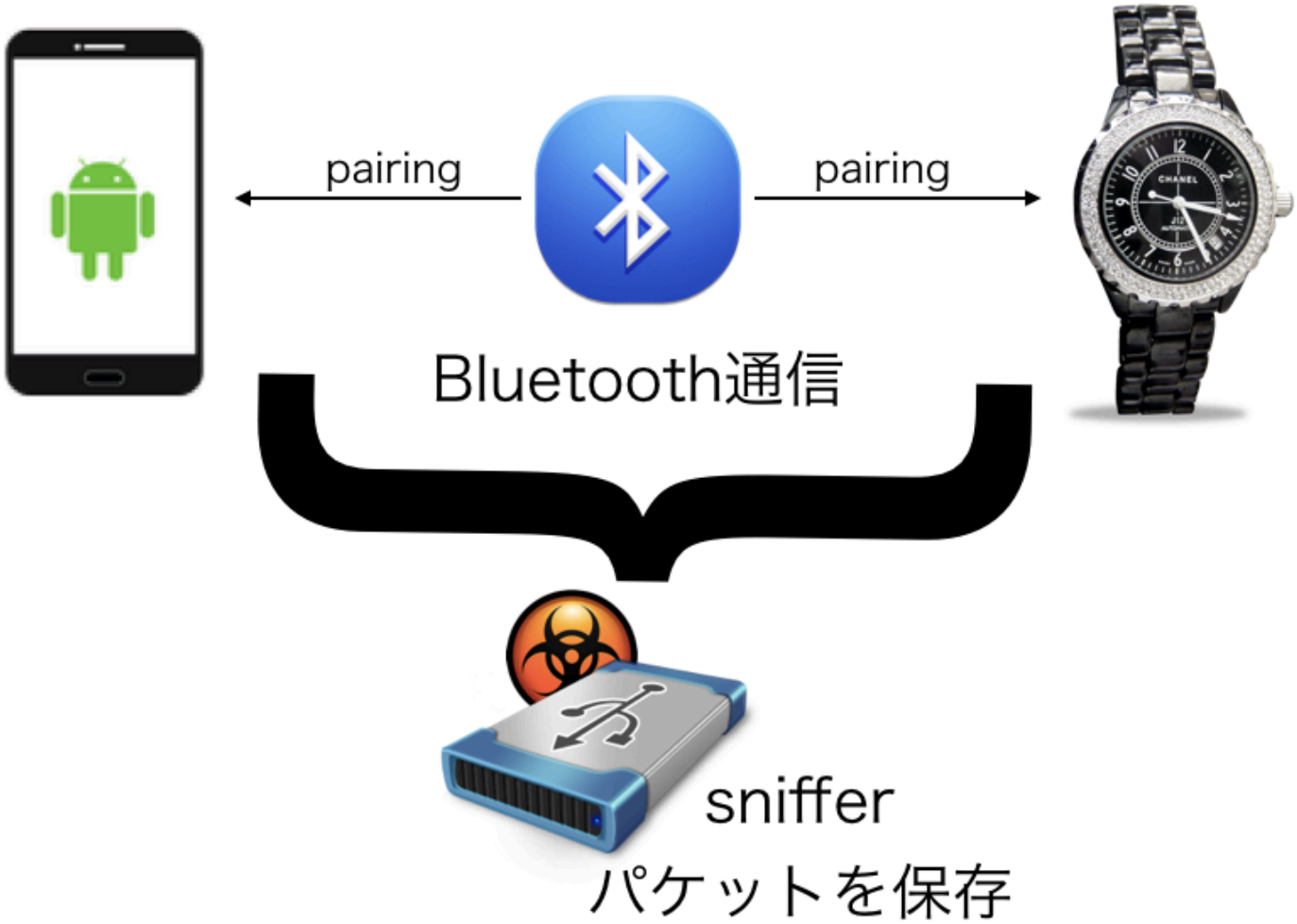
Android と通信して連絡帳などの同期

時計の概要

- アンドロイドに電話を掛ける命令をする
- アンドロイドから連絡帳を取得する
- OS が Android

以上の 3 点から、連絡帳データをパケットから抜き出してから UART の接続練習として基盤を取り出し物理接続を試みたいと思います。

パケットの取得は電球同様 Android の HCI スヌープログ機能で行います。
Bluefruit LE Sniffer で以下の様な構図で盗み取ろうと考える方もいると思います。



(図)スニッファを用いた通信 Bluetooth パケットの盗聴

もし、Bluefruit LE Sniffer がそこまで働いてくれればプライバシーなんて言葉は生まれないでしょう。安心してください、Bluefruit LE Sniffer が取れるのはせいぜい電球の value 値程度です。

それでは、時計とペアリングした状態で HCI スヌープログを ON にして連絡帳同期など行ってください。
おそらく連絡帳データがやりとりされパケットに保存されます。
筆者が提供したパケットデータにも偽物の連絡帳データが含まれています。
その連絡帳データを取り出してみましょう。
binwalk というファームウェア解析用のツールを使用します。
binwalk ではファームウェアに関わらずファイル内に含まれているファイルを発見次第抽出してくれる機能があります。
その機能は binwalk に"-eM"というオプションをつける事で利用可能です。
実際にそのオプションを使用して時計の通信パケットから連絡帳を抽出したのが下記の手順です。

```
#######################################################
root@root:~/time# binwalk -eM time.pcap
Scan Time:     2016-11-26 16:46:22
Target File:   /root/time/time.pcap
MD5 Checksum:  99b1fb850ed3a8545821cea930ddcf5b
Signatures:    344

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
17992         0x4648          XML document, version: "1.0"
27999         0x6D5F          XML document, version: "1.0"
47236         0xB884          XML document, version: "1.0"
#######################################################
```

この結果、ディレクトリに_time.pcap.extracted というディレクトリが生成されています。
そこには、先ほど binwalk が発見した 2 件の XML データが保存されています。

```
#######################################################
root@root:~/time# ls -l
total 52
-rw-r--r-- 1 root root 48698 Nov 26 16:45 time.pcap
drwxr-xr-x 2 root root  4096 Nov 26 16:46 _time.pcap.extracted

root@root:~/time# ls _time.pcap.extracted
4648.xml  6D5F.xml  B884.xml
#######################################################
```

実際に抽出したデータは strings コマンドなどで簡単に中身を表示する事が出来ます。

```
x-bt/vcard-listing
<?xml version="1.0"?><!DOCTYPE vcard-listing SYSTEM "vcard-listing.dtd"><vCard-listing version="1.0"><card ha
ndle="0.vcf" name="
"/><card handle="1.vcf" name="Tanaka"/><card handle="2.vcf" name="
"/></vCard-listing>
x-bt/vcard
sBEGIN:VCARD
VERSION:2.1
N:;Tanaka;;;
FN:Tanaka
X-PHONETIC-LAST-NAME:garou
TEL;CELL:08099996666
END:VCARD
ATD08099996666;
```

(図)抽出した連絡帳データ

筆者が時計の Bluetooth に gatttool で接続する前に基盤をむき出しにしてしまったため、gatttool で接続する事が出来ないのですが gatttool でペアリング認証する Bluetooth 機器にアクセスする場合は下記の様にします。

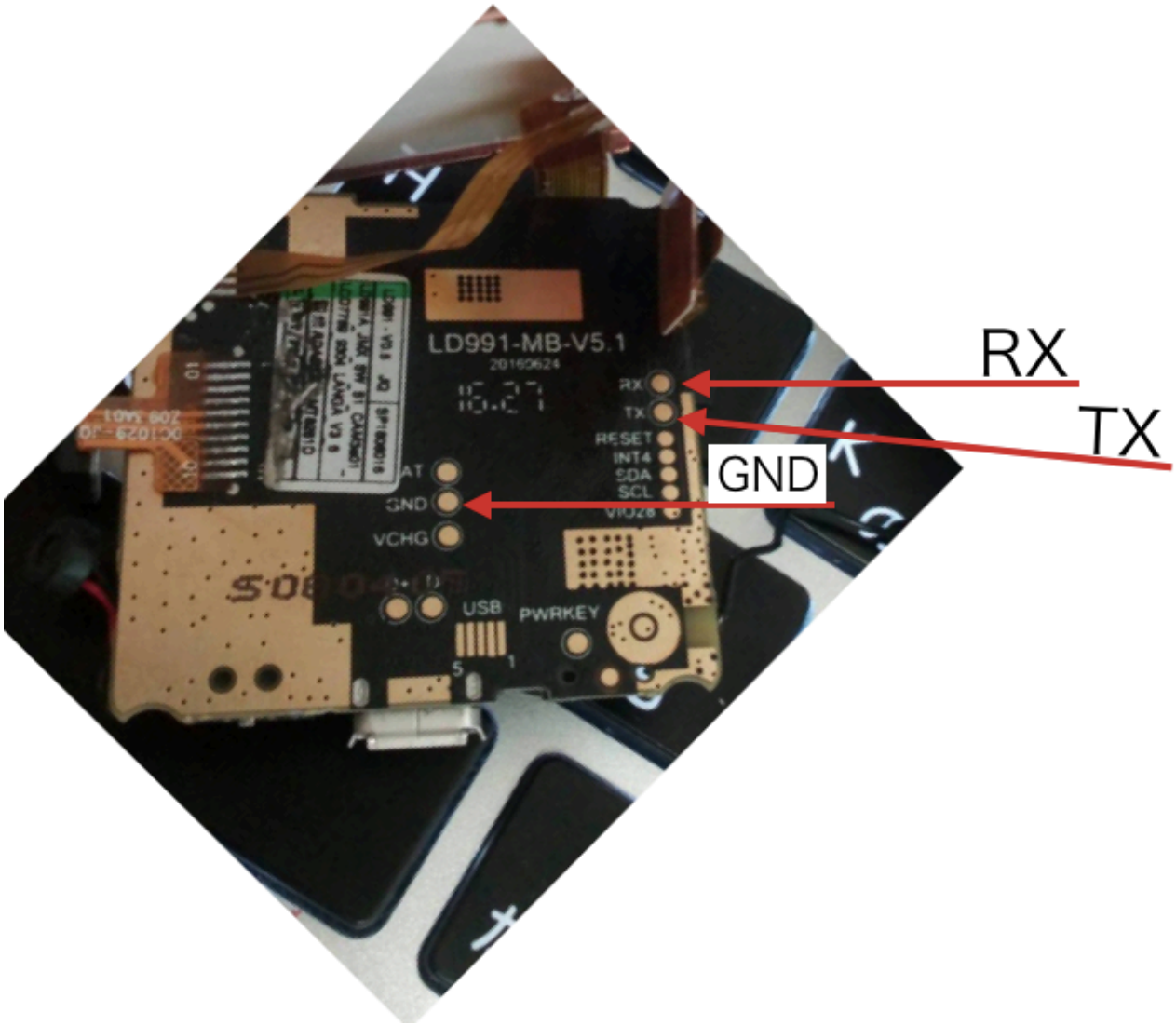
```
#########################################################
[x]ペアリング機器へのアクセス
gatttool --sec-level=high -t random -b <MAC Address> --primary
gatttool -l high -t random -b <MAC Address> --primary

--sec-level=medium
--sec-level=high
#########################################################
```

それでは、基盤のハッキングを行いましょう。
UART というシリアル転送方式のデータとパラレル転送方式のデータを相互に変換するためのデバイスを使用して時計に接続を試みます。
UART の越族では下記のピンが重要になります。
GND(基準電位との電位差が 0V)
TX-(プルアップ 0 ボルトから 3.3 ボルト変動)
Rx-(フローティング 3.3 ボルト,数百ミリボルト変動)

UART は、TX を RX に RX を TX に接続します。
GND ピンを GND ピンに接続
TX ピンを RX ピン接続
RX ピンを TX ピン接続

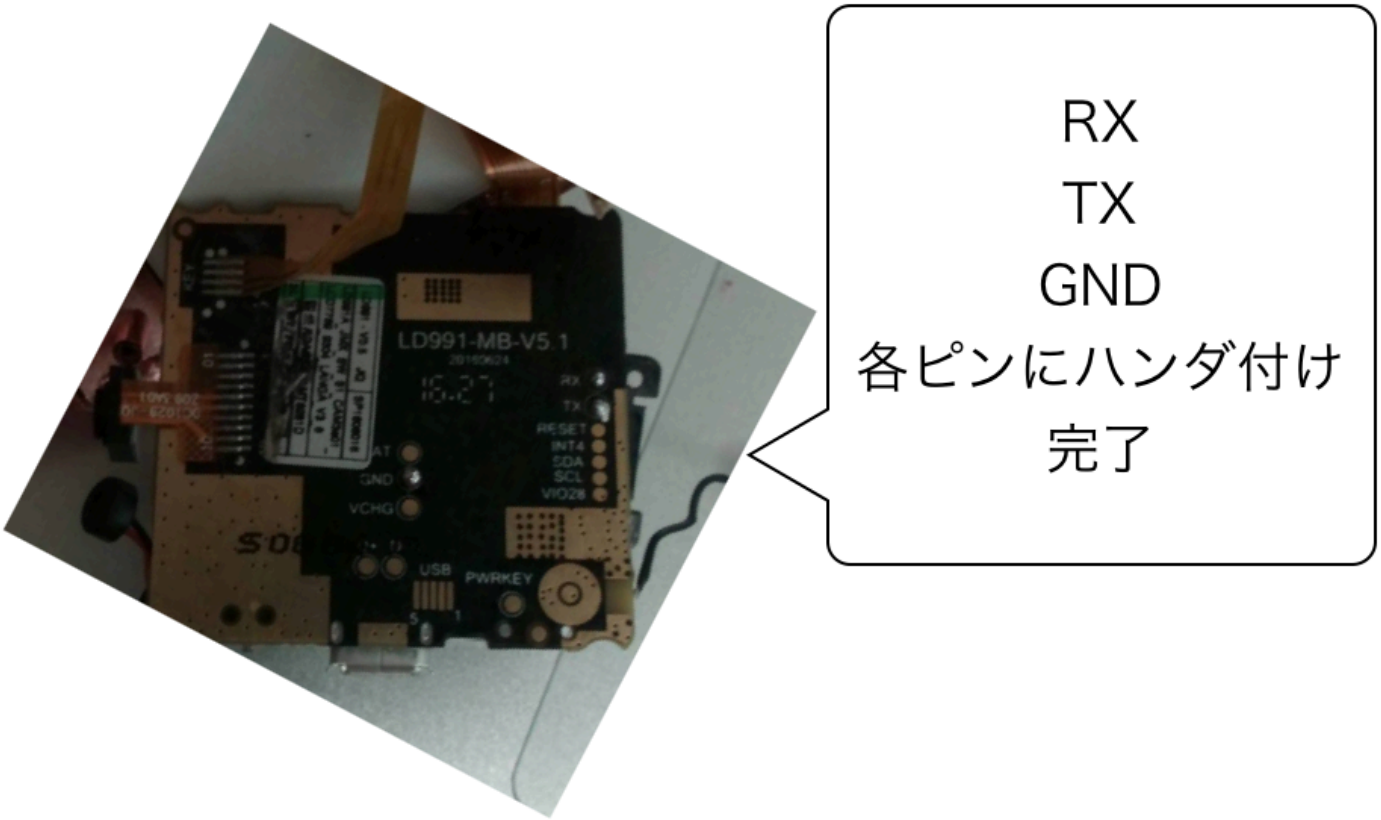
(図)UART の接続図

最初にケースなどを外し綺麗に基盤をむき出しにして、実際に各ピンがあるか確認しましょう。もし視認できない場合、自分で電圧を測定し調べるという行為をしなければなりません。



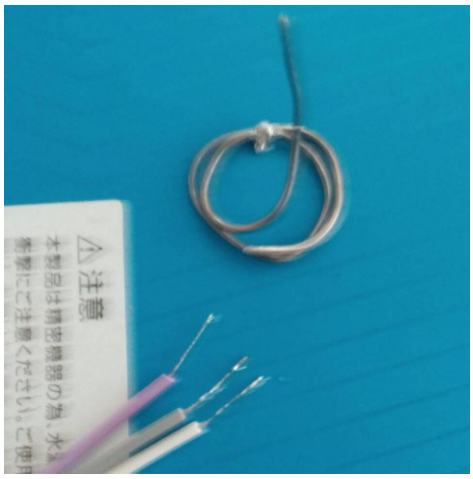
(図)時計の基盤から各ピンを探す

ピンが判別できたので各ピンにハンダをつけます。
最初に対象ピンにハンダをつけておく事で接続線のハンダ付けが楽になります。



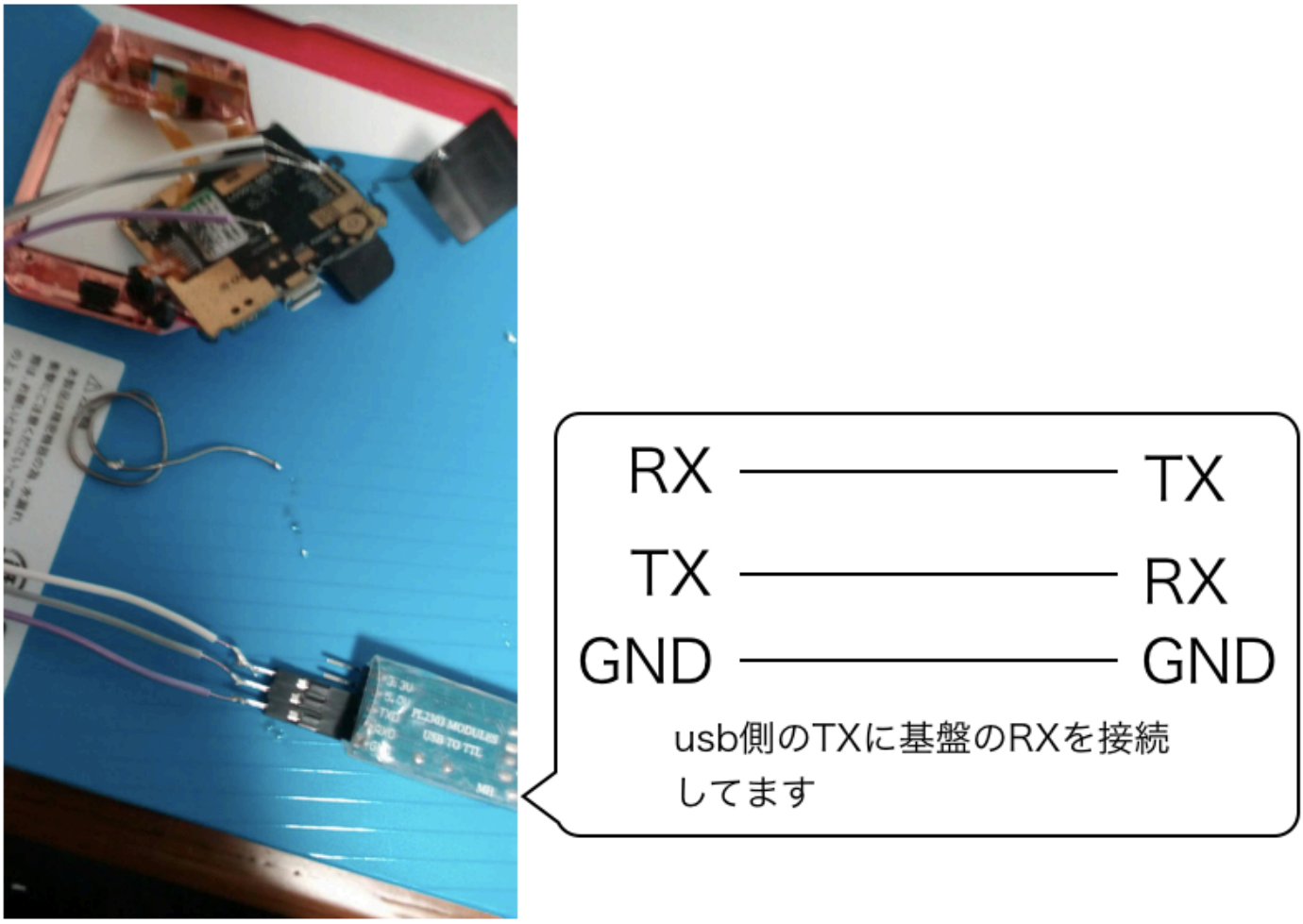
(図)先にハンダを対象ピンに塗りたくっとく

次に、接続線にハンダを塗りたくります。
接続線は一本一本の線の集まりで、すべての線を一つの方向にネジ回る事でバラバラの線を１つ束ねたりするのですが、その際に+a でハンダを塗っておくと接続線をピンとハンダ付けする時に凄くやりやすいです。



ハンダを丸めて戦闘を立たせてロウソクの様にして、ハンダを温めとかしつつ接続線にハンダをつけるのがコツです。
そうすると一人でもやりやすい。(基本作業は一人だと思いますが)

ここまで出来ればあとは各ピンに接続線をハンダ付けするだけになります。
各ピンに接続線をハンダ付けしたのが下記になります。



(図) UART モジュールシリアルコンバータに接続

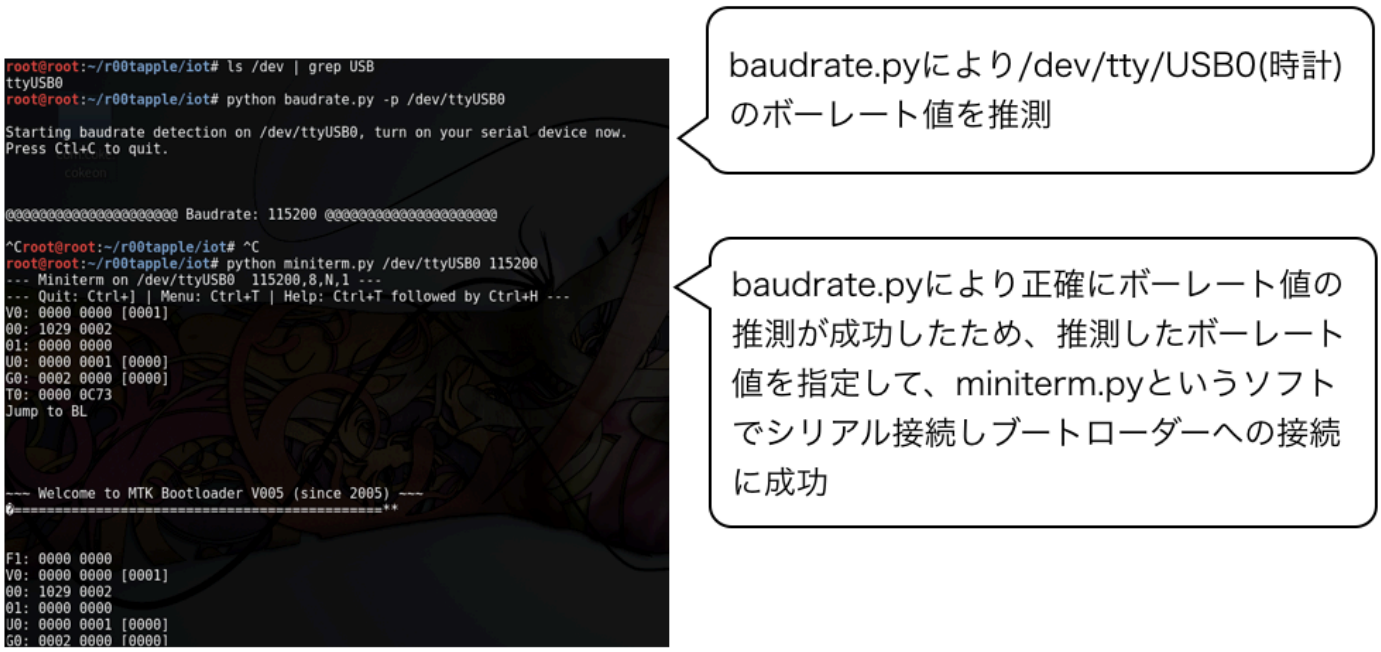
UART モジュールシリアルコンバータは、Google とか amazon で"uart usb"って検索すれば簡単に購入できます。

ここまで出来れば、推奨環境は Linux で私は手元の Kali Linux でボーレート値を測定します。
ボーレート値は、シリアル通信において転送速度を設定するパラメータとして” baud(ボー)” または” baudrate(ボーレート)” を単位として用いることがあり、単位は転送速度の単位である” bps” を用いることもあります。
UART の接続時に指定しなければならない値ですが、baudrate と bps は以下の違いがあります。

- baudrate:デジタルデータを 1 秒間に何回だけ変復調できるか示す値
- bps:1 秒間に転送することのできるデータ量を示す値

ボーレートを推測してくれる Python のオープンソースがあります。
そちらを使用すれば上手くいけば簡単に UART のシリアル接続をしてくれます。

baudrate.py が公開されている GITHUB(ボーレート推測用)
https://github.com/719Ben/Baudrate.py/blob/master/baudrate.py
miniterm.py が公開されている GITHUB(シリアル接続用)
https://github.com/pyserial/pyserial/blob/master/serial/tools/miniterm.py



(図)UART 接続によるブートローダーへのアクセス

上記の様にボーレートの推測さえ成功すれば簡単に接続する事ができます。
ボーレートが正しく無い場合も接続出来ますが、文字化けしてしまいます。
CISCO のルーターなどもボーレートが違うと文字化けしてしまいます、それと同じ問題です。
時にこの様な解析を防ぐためかボーレートが曖昧な機器がありますが、その場合ボーレートの推測な難儀になりロジックアナライザなどでプロトコルダンプし調べる事が一番確実な手段になるでしょう。

ROOTAPPLE

自動販売機のハッキング

■自動販売機のハッキング

最後に自動販売機のハッキングについて説明します。
自動販売機のハッキングと言っても不正にコカコーラを盗み出す手法ではありません。
自動販売機内にある Bluetooth に対して任意の動作をするための方法になります。

自動販売機のハッキング概要

- primary 情報により自動販売機に割り当てられた機器名の取得
- アプリ改ざんによる任意の番号表示

電球のハッキングを学びました。
実は自動販売機のハッキングと電球のハッキングは大差がありません。
要するに、電球のハッキングを理解すればある程度の Bluetooth 機器はハッキングできるのです。
自動販売機の MAC アドレスを調べてみましょう。
自動販売機の目の前まで行って以下のコマンドを入力しましょう。

```
root@root:/var/www/html/input-output# hcitool lescan
LE Scan ...
00:AE:FA:BA:82:6C (unknown)
^C
```

(図)hcitool による自動販売機の MAC アドレスの検索

MAC アドレスを無事に取得できたので、これに gatttool で接続しましょう。
以下は、gatttoool で接続して primary の漏洩までさせる流れになります。

```
root@root:/var/www/html/input-output# gatttool -i hci1 -b 00:AE:FA:BA:82:6C -I
[00:AE:FA:BA:82:6C][LE]> connect
Attempting to connect to 00:AE:FA:BA:82:6C
Connection successful
[00:AE:FA:BA:82:6C][LE]> primary
attr handle: 0x0001, end grp handle: 0x0009 uuid: 00001800-0000-1000-8000-00805f9b34fb
attr handle: 0x000c, end grp handle: 0x000f uuid: 00001801-0000-1000-8000-00805f9b34fb
attr handle: 0x0010, end grp handle: 0x0014 uuid: 0a37b48b-bc03-42f6-972f-34a3259da6ef
attr handle: 0x0015, end grp handle: 0x001d uuid: 688f48ca-8f67-41a6-b05f-9b99f3442ff4
attr handle: 0x001e, end grp handle: 0x0026 uuid: d891b009-be19-4b54-9d64-384c875c4aa1
attr handle: 0x0027, end grp handle: 0x0034 uuid: 5b74eab6-f06b-4d84-b85f-e90af679e6ed
attr handle: 0x0035, end grp handle: 0x0037 uuid: bcd01923-4eda-4e69-a1f4-5755e19e1a73
```

(図)primary 情報の漏洩

電球で説明した様に、UUID の 1800 は Generic Access で Bluetooth 機器名を取得できたはずです。

情報を引き出す時は、write では無く read ですね。
gattttool による Generic Access の情報表示は以下の様に行います。
Generic Access の機器名に割り当てられた 16 ビットの値は 0x2a で下記の値です。
[+]andle: 0x0003, uuid: 0000**2a00**-0000-1000-8000-00805f9b34fb
この値を char-desc コマンドで以下の通りに打ちます。

$char-desc 0x0001-0x0009

この時に指定している、0x0001 と 0x0009 は primary 情報で UUID の 1800 が以下の様に、
attr **handle: 0x0001**, end grp **handle: 0x0009** uuid: 00001800-0000-1000-8000-00805f9b34fb
handle が 0x0001 から始まり 0x0009 で終わっているからでしたね。

```
root@root:/var/www/html/input-output# gatttool -i hci1 -b 00:AE:FA:BA:82:6C -I
[00:AE:FA:BA:82:6C][LE]> connect
Attempting to connect to 00:AE:FA:BA:82:6C
Connection successful
[00:AE:FA:BA:82:6C][LE]> char-desc 0x0001 0x0009
handle: 0x0001, uuid: 00002800-0000-1000-8000-00805f9b34fb
handle: 0x0002, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0003, uuid: 00002a00-0000-1000-8000-00805f9b34fb
handle: 0x0004, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0005, uuid: 00002a01-0000-1000-8000-00805f9b34fb
handle: 0x0006, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0007, uuid: 00002a02-0000-1000-8000-00805f9b34fb
handle: 0x0008, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0009, uuid: 00002a04-0000-1000-8000-00805f9b34fb
[00:AE:FA:BA:82:6C][LE]> char-read-hnd 0x0003
Characteristic value/descriptor: 4b 4f 48 41 38 32 36 43
```

(図)Generic Access で 0x2a(機器名)の表示

これにより、機器名が 16 進数で表示されました。
Characteristic value/descriptor: 4b 4f 48 41 38 32 36 43
これを変換すると、**KH86** ですね。
今回の自動販売機の機器名は KH86 であるとわかります。
最後に primary 情報をすべて表示して終わります。

##
root@root:/var/www/html/input-output# gatttool -i hci1 -b 00:AE:FA:BA:82:6C -I
[00:AE:FA:BA:82:6C][LE]> connect
Attempting to connect to 00:AE:FA:BA:82:6C
Connection successful
[00:AE:FA:BA:82:6C][LE]> char-read-hnd 0x0001
Characteristic value/descriptor: 00 18

[00:AE:FA:BA:82:6C][LE]> char-read-hnd 0x0002
Characteristic value/descriptor: 0a 03 00 00 2a
[00:AE:FA:BA:82:6C][LE]> char-read-hnd 0x0003
Characteristic value/descriptor: 4b 4f 48 41 38 32 36 43
[00:AE:FA:BA:82:6C][LE]> char-read-hnd 0x0004
Characteristic value/descriptor: 0a 05 00 01 2a
[00:AE:FA:BA:82:6C][LE]> char-read-hnd 0x0005
Characteristic value/descriptor: 00 00
[00:AE:FA:BA:82:6C][LE]> char-read-hnd 0x0006
Characteristic value/descriptor: 0a 07 00 02 2a
[00:AE:FA:BA:82:6C][LE]> char-read-hnd 0x0007
Characteristic value/descriptor: 00
[00:AE:FA:BA:82:6C][LE]> char-read-hnd 0x0008
Characteristic value/descriptor: 02 09 00 04 2a
[00:AE:FA:BA:82:6C][LE]> char-read-hnd 0x0009
Characteristic value/descriptor: 08 00 10 00 00 00 64 00
##

他にもバッテリ内臓であれば primary から調べる事が可能です、primary 値がどの様な値になっているかは是非調べてアウトプットしてください。
逆に primary 情報からどの様にハッキングに展開するか考えるのも面白いですね。
みなさんのアウトプットを楽しみにしています。

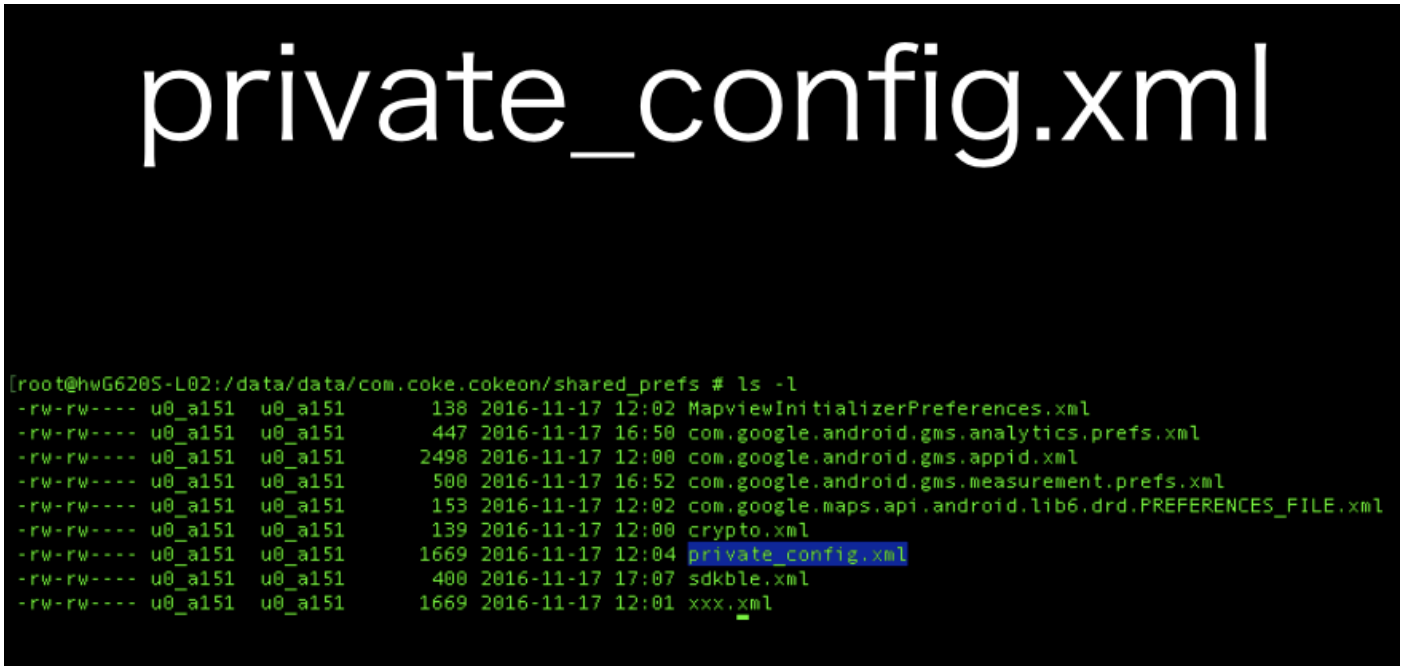
それでは、最後に一番面白い任意の番号表示を説明します。

アプリの改ざんにより与えられたランダム数字では無く任意の数字を表示させた。

(図)自動販売機に 77777 を表示させた

これは簡単です。
しかし、これからの IoT のハッキングで重要になるでしょう。
私が 77777 や 66666 を gatttool により表示させたのでは無くアプリを改ざんした理由はそっちの方が簡単だからでした。
しかし、自動車(tesla)のハッキングをした技術では Android に格納される Oauth などの情報を盗み出すマルウェアを観戦させアプリの仕様を逆手に取り不正に Tesla 車をジャックするというハッキングがありまう。
面白いですね。それは Android Malware Used to Hack and Steal a Tesla Card で検索すれば色々情報が出てきます。

Android の apk のデータにアクセスするためにまず root 化しました。
root 化する事で sqlite にも簡単に簡単にアクセス出来ます。
そして対象アプリのディレクトリを調べました。



(図)アプリ内の重要なファイル

ディレクトリ内を調べる事で、private_config.xml など幾つか重要なデータ(他にはポイントカードの値らしきものなど)を見つけました。

この private_config.xml は以下の様な値です。(書き換えた情報で申し訳ないですが)

```
#######################################################
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
```

```
<map>
    <string name="vmConnectionId">77777</string>
    <boolean name="hasFinishedSetup" value="true" />
    <int name="stampNumber" value="0" />
    <string name="appId">Iam@<hr>apple</string>
    <string
name="accessToken">AQFfmmzuVI+o23n8jzc/lvxU5iuteQxVJaCVazFncWK8Ea8skOQFhUHVcqG
vRphFK/XcwgGCuRl3
vvKZQvP61oMDnPtMpL7HS3YPji4dZQZGLZpkfljJivmBHzKE5i978HBIzmUEgZNXwg6SXFcfWIMe
I5QQSZMTsmoqUFCyTFB4A0KidFI9vmgbqHznti/8YjZMpZ0BaeanFMoNCCs=
    </string>
    <string
name="registrationId">AQHGUWglmhYRmjbff9z3q77lHeS5RFJ1sEJHjsp8lUgQO9xgFt/N2v/+
rLbCJ5U4QEDm/Py99DKG
HDL92TwDrY6pmZR9RO3DTtOAXD7+3ZpJqh6y0lcCxB0eT1sMWmbcR3lbibj6NcScT4SXx23SwgEk
2MVoQqABgCDzcqvQXafjYmgFR7jixO0qG7SzC1AQuwgsY9xf/mInILv+o0AYSjTjbgqLZjTq49oM
txWz6lgoSNvTU6arJgdlJS32j2/wSde/32omvxCBCYwqBU4sD3pOKzBfHd/uJ4e4fcDPtpLsYHoc
VbUab6Zd
    </string>
    <string
name="refreshToken">AQH/tSkM+M2wFEf2ZteybzwNehVnAE3OdJalvfszkSn0a7b3y2GappDORV
hgbxfc8kCw0MqKsTTz
1clSJYD9RjpeeGbww76biqVnKC1kJfGCvu5XGCoWKoTldZqQwV9qH1wgPUpbtgzEwZMYHn8q3Y2v
z4wxJZYy6jzIhdrqXHl8yRh8K3MMbDPXLEHqQJ3JYZvjd1OtWqs8AH0wiKk=
    </string>
    <string name="appPin"><hr>Bye!</string>
    <boolean name="isChildUnderTwelve" value="true" />
    <boolean name="newlyArrivedMark" value="true" />
    <string name="GALineAddFriendEventLabel">important</string>
    <boolean name="restartSDKManager" value="true" />
    <string name="GALoginEventLabel">important</string>
    <string name="latestNewsPublished">2016/11/16 10:00</string>
    <string
name="hardwareNo">AQHU+qBH0pSRvDvIOqaVPl5J5NRUOckNADoVO56K0nFEpdKQtM9hRzljttoP
0VtmXkUM8qqZ65W9
4WxsSjY=
    </string>
</map>
```

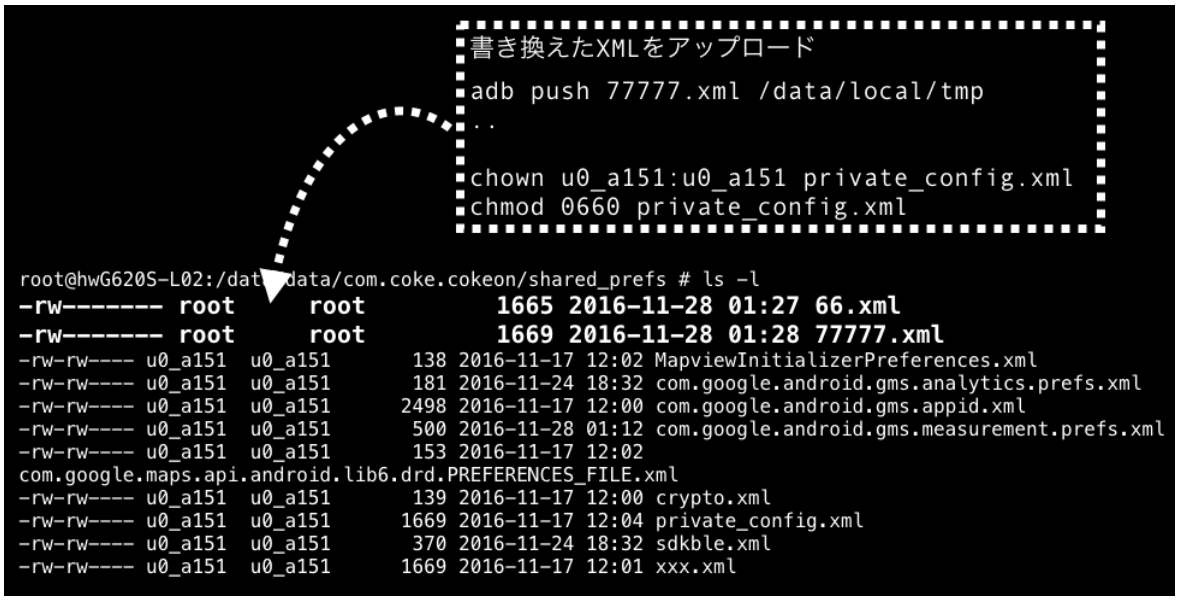
##

この時、一目で書き換えるポイントがわかります。

```
<string name="vmConnectionId">77777</string>
```

ここを書き換えれば任意の数値を表示できるはずです。(他の重要なデータを書き換えた不整合にした場合を除いてですが)

以下の様に書き換えます。



(図)private_config.xml を書き換える

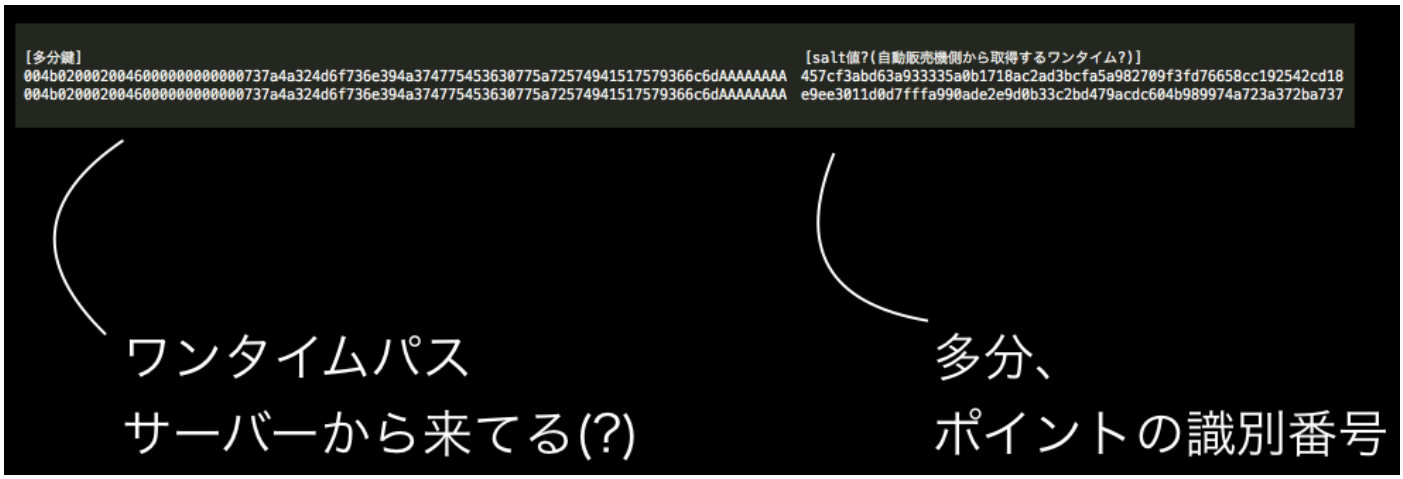
実行権限や所有者の情報を他のファイルに合わせます。

これで整合性は保たれるのでアプリが勝手に読み込み自動販売機に送信してくれるはずです。

勿論この値を書き換え、Android のアプリケーションを再起動し自動販売機に接続する事で任意の数値を表示する事に成功しました。

(図)自動販売機のやり取り

自動販売機のチェックは厳しかった印象です、時間がゆるす限りにハックはしましたがすべてを理解していないので概要図になります。
4G・WiFi の通信をうまくハッキングできればより面白い結果になるかと思いますが、この様に IoT というのは単純な動作しかしません。
Android という攻撃者が不正な値に改ざんした情報を送ればそれを信用し取り扱ってしまいます。
そのため、サーバーのチェックは欠かせません。
自動販売機はポイントシステムとして動作しますが、その際に送信しているデータは下記の様になります。



(図)自動販売機に送信しているデータ

私が考えるに、このワンタイムパスワードが重要なポイントでこのワンタイムパスワードはサーバー側が機器固有情報などをもとに生成します。
そして生成されたワンイタイムパスにポイント番号を付加し自動販売機で購入決済された瞬間
[ワんタイムパスワード] + [ポイント番号] + [購入後の値]
を返すのではないかと思います。
その情報をもとに何かしらの手順でサーバーがチェックを行い、ポイントがサーバー内のデータベースに正しく付加されればアプリ内の XML 内の stampNumber という箇所を書き換えローカル上でも現在のポイント数が確認できる様になるのではち思います。

```
<int name="stampNumber" value="0" />
```
※private_config.xml 内にあるスタンプ数情報を保存する箇所

以上で簡単ですが、自動販売機のハッキングになります。
これから IoT といえど以下に正規のアプリを改ざんしハッキングするかが問われるのかと思います。gatttool とアプリのハッキング両者をうまく使う事で Bluetooth のハッキングは上手く簡単にできる様になりそうですね。