Hacking BLE Smartwatch

@smrx86
Independent Security Researcher.

Abstrak—Tulisan ini ditujukan untuk menganalisa dan memahami prosedur authentifikasi jam pintar dengan perangkat telepon pintar berbasis android di frekuensi 2,4 Ghz berbasis Bluetooth Low Energy. Berangkat dari analisa proses authentifikasi ini penulis akan coba memaparkan bagaimana fitur keamanan tersebut dapat diemulasikan lewat perangkat lain**.**

Kata kunci— *Reverse engineering, Android, IoT Security, BLE***.**

1. Pendahuluan

Bluetooth Low Energy (BLE) diperkenalkan pada tahun 2010 sebagai standar baru penggunaan komunikasi via Bluetooth versi 4.0. BLE ini di gadang gadang sebagai sebagai penggunaan frekuensi Bluetooth dengan konsumsi daya yang sangat rendah. Selain itu juga diproduksi dengan harga yang murah dan ukuran yang kecil.

Faktor-faktor keunggulan tersebut diatas diharapkan bisa menurukan biaya produksi serta memperbanyak varian produk yang dapat terhubung dengan perangkat pintar konsumen.

Disisi lain keunggulan BLE ini tidak didukung dengan faktor keamanan dalam penerapannya yang tidak terlalu bagus. Data yang dikirmkan oleh perangkat dapat mudah di intip dan di terjemahkan jika tidak menerapkan enkripsi bagus dalam komunikasinya. Firmware yang berada dibelakang yang mengatur processing data memegang penting dalam proses authentifikasi serta enkripsi data yang dikirimkan setelahnya.

Dibanyak konfrensi keamanan IT telah banyak di publikasikan tentang produk perangkat IoT yang tidak menerapkan standar keamanan yang baik sehingga banyak ditemukan celah kritikal yang mengancam penggunanya.

Disini penulis mencoba memperlihatkan bagaimana proses authentifikasi di perangkat jam Amazfit Bip dengan aplikasi pendukungnya di perangkat telpon pintar berbasis Android.

2. Tulisan Rujukan

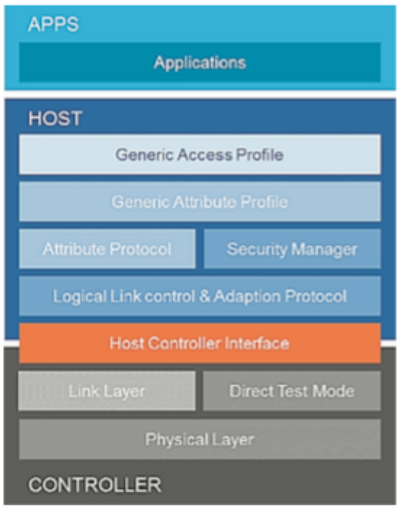
Ada beberapa tulisan yang menjadi rujukan dalam penulisan paper ini. Namun artikel paling berpengaruh dan menjadi titik tolak adalah tulisan Leo Soares di dalam blognya. Dalam postingannya itu Leo Soares memetakan proses authentifikasi di gelang pintar Miband 2 yang hampir sama persis di gunakan di Amazfit Bip.[1]

Selain itu artikel Andrey Nikishaev cukup baik mengulas bagaimana Miband2 berinteraksi dengan applikasi Mifit bermodalkan Bluetooth scanner, gattools dan btsnoop_hci.log.[2]

Tulisan lain yang cukup penting adalah tulisan David Lodge yang dengan jelas mengulas tentang bagaimana menggunakan frida untuk melihat ATT command yang dikirimkan ke Bluetooth amplifier.[3]

3. Konsep Bluetooth Low Energy

BLE sebagai sarana komunikasi data, secara umum terdiri dari 3 layer; Controller, Host, Application.



Gambar 1. BLE stack protocol

Proses pairing dilakukan dengan perintah yang diberikan oleh aplikasi, kemudian di teruskan ke layer host yang diformulasikan ke dalam perintah generic attribute profile (GATT), kemudian dipecah kedalam perintah yang lebih spesifik ke attribute profile (ATT). Alur akhir perintah akan diteruskan controller di Host Controller Interface (HCI). Perangkat yang terdaftar sebagai HCI dapat diketahui dengan perintah *hciconfig* di terminal command.

4. Metodologi

Dalam tulisan ini metode yang digunakan adalah metode whitebox. Dimana penulis menempatkan diri sebagai pengguna perangkat yang sudah yang punya otoritas penuh untuk control perangkat, mulai dari authentifikasi awal hingga mencoba semua fungsi aplikasi yang ada di Amazfit Bip maupun perangkat yang terhubung dengannya.

Di tulisan-tulisan lain sebenarnya sudah banyak metode pengujian blackbox untuk analisa komunikasi BLE.[4] Namun sejauh percobaan penulis, metode blackbox yang umum di gunakan

---

[1] Leo Soares. "Mi Band 2, Part 1: Authentication.", Internet: https://leojrfs.github.io/writing/miband2-part1-auth/, Nov. 25, 2017 [Feb 24, 2019].

[2] Andrey Nikishaev. "How I hacked my Xiaomi MiBand 2 fitness tracker—a step-by-step Linux guide", Internet: https://medium.com/machine-learning-world/how-i-hacked-xiaomi-miband-2-to-control-it-from-linux-a5bd2f36d3ad, Mar. 26, 2018 [Feb 24, 2019].

[3] David Lodge, "Reverse Engineering BLE from Android apps with Frida", Internet: https://www.pentestpartners.com/security-blog/reverse-engineering-ble-from-android-apps-with-frida/, Feb 23, 2018 [Feb 24, 2019].

[4] Anand, "Exploiting Bluetooth Low Energy using Gattacker for IoT - Step-by-Step Guide", Internet: https://blog.attify.com/hacking-bluetooth-low-energy/ ,Mar 01, 2018 [Feb 24, 2019].

kurang efektif untuk memeriksa aliran data yang dikirim maupun diterima.

Gambar 2. Gattacker gagal dalam metode MITM

Dugaan kuat kalau modul bleno dan noble dalam aplikasi gattacker tidak dapat menangani proses authentifikasi yang berjalan dengan tempo yang sangat singkat. Impactnya perangkat akan otomatis akan memutuskan perangkat yang dalam kondisi pairing.[5]

Selain itu juga ada metode sniffing dengan dongle seperti ubertooth dll, akan tetapi tidak ada jaminan ATT command yang dicapture akan berupa data yang plain.

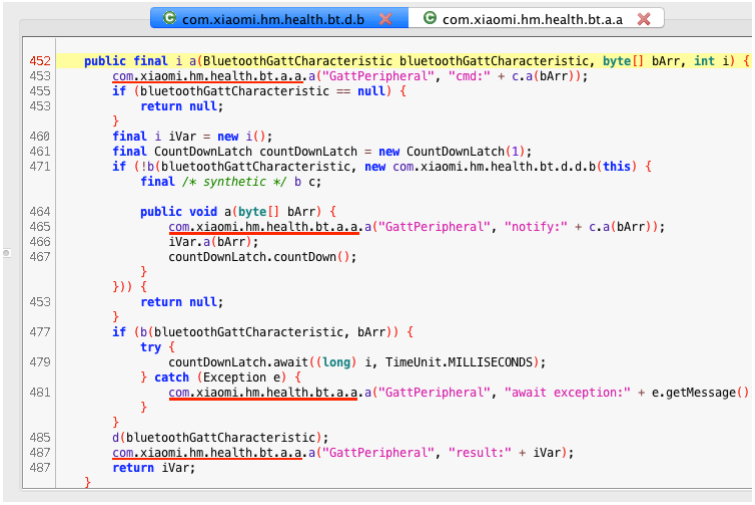
Dengan kondisi seperti ini, cara yang paling tepat adalah dengan memantau data yang dikirimkan aplikasi dengan bantuan frida dan menvalidasi nilai datanya dengan log Bluetooth yang tersimpan di penyimpanan lokal Android (btsnoop_hci.log).

4.a. Debugging dengan FRIDA.

Frida adalah framework yang memungkinkan kita untuk melakukan tracing, profiling dan debugging apa saja yang sedang dieksekusi oleh sebuah applikasi. Kapasitas inilah yang kemudian dikenal dengan instrumentation.

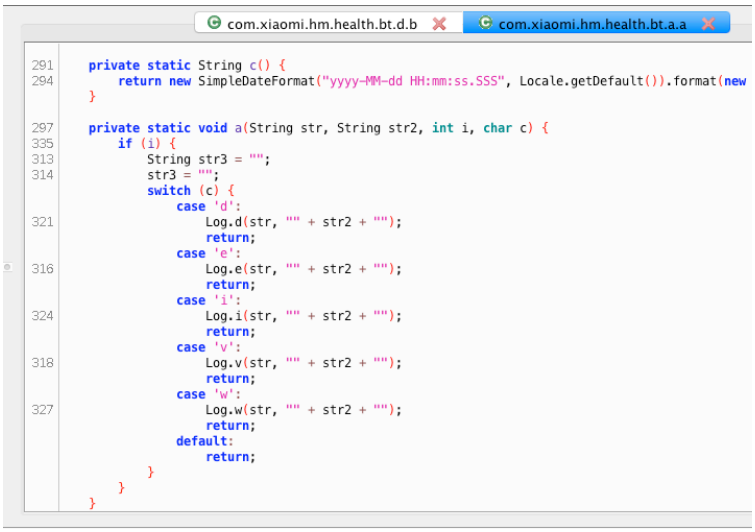
Fungsi frida disini saya gunakan untuk memudahkan proses debug dan melakukan logging pada java class yang memanggil modul BluetoothGATT didalam source code applikasi Mi-Fit.

Dari penelusuran menggunakan JADX diketahui bahwa class *com.xiaomi.hm.health.bt.d.b* memanggil modul android bluetoothgatt dan melemparkan hasilnya ke *com.xiaomi.hm.health.bt.a.a* di method *a*.



Gambar 3. Class yang memanggil bluetoothgatt.

Class *com.xiaomi.hm.health.bt.a.a* sendiri dapat kita lihat kalau method *a* adalah method yang menghandle logging debug, error hingga logging void.



Gambar 4. method a menghandle logging

Di aplikasi android tidak semua jenis logging bisa muncul di *logcat*. Dalam best practice keamanan aplikasi, log yang menampung konten kredensial tidak boleh berada pada logging jenis log.d dan log.v.

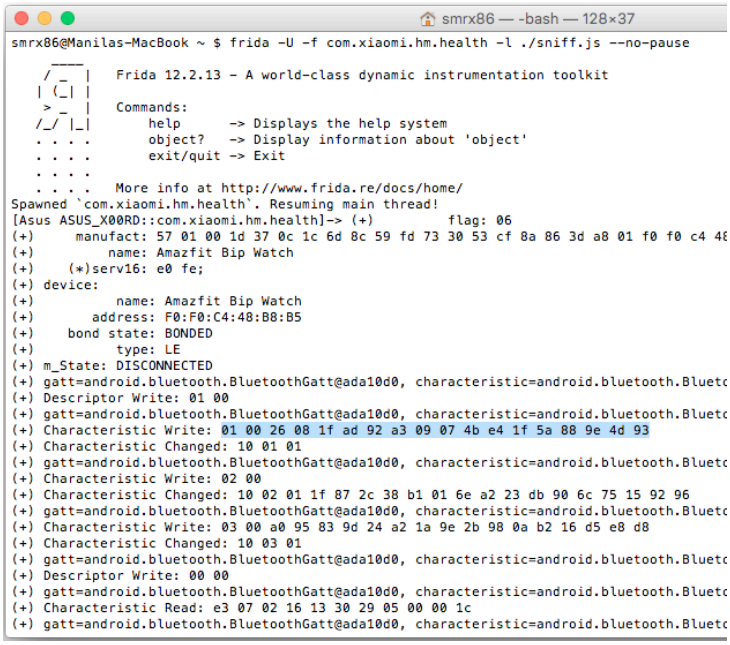
Untuk mengambil data bluetoothgatt dari *com.xiaomi.hm.health.bt.a.a* kita dapat membuat javascript frida dengan format sebagai berikut:

```
Java.perform(function() {

    var ble = Java.use("com.xiaomi.hm.health.bt.a.a");
    var sniff = ble.a.overload('java.lang.String');

    sniff.implementation = function (data) {
        console.log("(+) "+ data);
    }
});
```

Gambar 5. Script sniff.js untuk logging *com.xiaomi.hm.health.bt.a.a*

[5] Sławomir Jasek, "Hacking Bluetooth Smart Locks - workshop", Internet: https://smartlockpicking.com/slides/BruCON0x09_2017_Hacking_Bluetooth_Smart_locks.pdf, Oct 05, 2017 [Feb 24, 2019].

Setelah di test akan di dapatkan hasil logging yang dikirimkan dan diterima oleh aplikasi,



Gambar 6. Hasil sniff.js saat dijalankan.

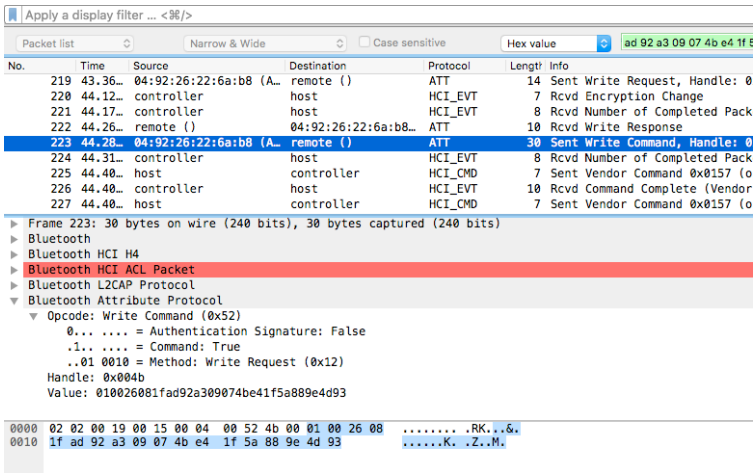
terlihat string "010026081fad92a309074be41f5a889e4d93" yang merupakan key pertama yang dikirimkan oleh aplikasi ke AmazfitBip.

4.b. Logging di btsnoop_hci.log

Menemukan data logging Bluetooth lewat btsnoop_hci.log adalah cara umum paling mudah untuk mengetahui value, handle serta characteristic profile yang bertanggung jawab didalamnya.

Dengan mengaktifkan bluetooth snooping lewat developer option, kita dapat mengambil logging di direktori */data/misc/bluetooth/logs/btsnoop_hci.log*.

File logging yang diambil bisa kemudian bisa dibaca dengan wireshark,

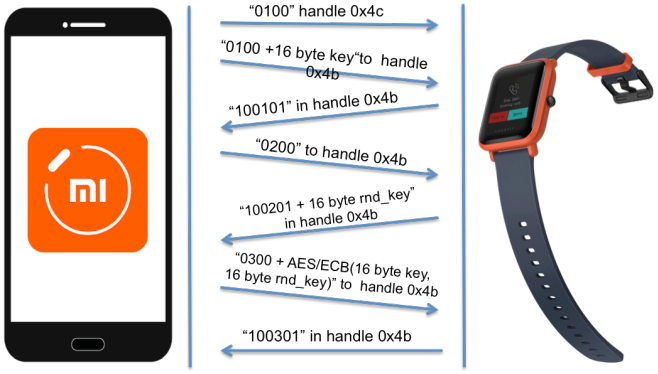


Gambar 7. btsnoop_hci.log yang dibuka dengan wireshark.

Dari log file yang dibuka dengan wireshark dapat diketahui jika ATT string "010026081fad92a309074be41f5a889e4d93" yang dikirimkan perangkat merupakan *char_write* ke *handle 0x004b.*

5. Analisa Prosedur Authentifikasi

Dari hasil logging frida yang divalidasi dengan btsnoop_hci.log kita bisa mengetahui alurnya,



- Pada ketukan awal interaksi, aplikasi akan mengirimkan perintah char_write "0100" ke handle *0x4c* sebagai notifikasi bahwa authentifikasi akan dimulai,
- Setelah itu baru "0100 +**16 byte key**" dikirimkan aplikasi ke handle *0x4b*.
- Kemudian Amazfitbip akan menginformasikan jika nilai handle 0x4b saat ini adalah "100101" sebagai tanda bagi aplikasi untuk mengirimkan random key.
- Aplikasi akan mengirikan notifikasi meminta random key dengan nilai "0200" ke handle *0x4b*.
- AmazfitBip mengirimkan "100201 + **16 byte random key**)" ke applikasi lewat handle *0x4b*.
- Disini aplikasi akan meresponse dengan kalkulasi "0300 + **AES/ECB/NoPadding(16 byte key, 16 byte random key)**" ke handle *0x4b***.**

- AmazfitBip akan menvalidasi nilai tersebut, jika sesuai maka akan di informasikan nilai "100301" sebagai notifikasi kalau akses aplikasi sudah terauntentifikasi.

6. Eksploitasi

Pada tahap eksploitasi saya menggunakan script yang telah ditulis oleh Volodymyr Shymanskyy dengan penyusuaian,[6]

- b'/x01/x00' di baris _send_key_cmd.
- b'/x02/x00' di baris _send_rnd_cmd.
- b'/x03/x00' di baris _send_enc_cmd

Percobaan eksekusi script dengan menggunakan linux debian yang terhubung ke dongle usb bluetooth CSR 4.0 mendapatkan hasil bahwa proses authentifikasi ke AmazfitBip dapat di emulasikan dengan perangkat lain jika mampu menformulasikan prosedur yang benar dan tepat.

Gambar 8. script bip.py sukses mengirimkan notifikasi ke AmazfitBip.

Proses authentifikasi inilah yang kemudian menjadi titik tolak dari project gadgetbridge dalam pengembangan aplikasi yang mendukung jam pintar AmazfitBip untuk bisa terkoneksi dengannya.[7]

7. Referensi

1. Leo Soares. "Mi Band 2, Part 1: Authentication.", Internet: https://leojrfs.github.io/writing/miband2-part1-auth/, Nov. 25, 2017 [Feb 24, 2019].
2. Andrey Nikishaev. "How I hacked my Xiaomi MiBand 2 fitness tracker—a step-by-step Linux guide", Internet: https://medium.com/machine-learning-world/how-i-hacked-xiaomi-miband-2-to-control-it-from-linux-a5bd2f36d3ad, Mar. 26, 2018 [Feb 24, 2019].
3. David Lodge, "Reverse Engineering BLE from Android apps with Frida", Internet: https://www.pentestpartners.com/security-blog/reverse-engineering-ble-from-android-apps-with-frida/, Feb 23, 2018 [Feb 24, 2019].
4. Anand, "Exploiting Bluetooth Low Energy using Gattacker for IoT - Step-by-Step Guide", Internet: https://blog.attify.com/hacking-bluetooth-low-energy/ ,Mar 01, 2018 [Feb 24, 2019].
5. Sławomir Jasek, "Hacking Bluetooth Smart Locks - workshop", Internet: https://smartlockpicking.com/slides/BruCON0x09_2017_Hacking_Bluetooth_Smart_locks.pdf, Oct 05, 2017 [Feb 24, 2019].
6. Volodymyr Shymanskyy. "Miband2.py", Internet: https://github.com/vshymanskyy/miband2-python-test/raw/master/miband2.py, Mar. 17, 2018 [Feb 24, 2019].
7. Gadgetbridge. "Amazfit Bip Status", Internet: https://github.com/Freeyourgadget/Gadgetbridge/wiki/Amazfit-Bip#amazfit-bip-status, Jan. 19, 2019 [Feb 24, 2019].

[6] Volodymyr Shymanskyy. "Miband2.py", Internet: https://github.com/vshymanskyy/miband2-python-test/raw/master/miband2.py, Mar. 17, 2018 [Feb 24, 2019].

[7] Gadgetbridge. "Amazfit Bip Status", Internet: https://github.com/Freeyourgadget/Gadgetbridge/wiki/Amazfit-Bip#amazfit-bip-status, Jan. 19, 2019 [Feb 24, 2019].

Lampiran

```
========================================== sniff.js ==========================================
Java.perform(function() {

    var ble = Java.use("com.xiaomi.hm.health.bt.a.a");
    var sniff = ble.a.overload('java.lang.String');

    sniff.implementation = function (data) {
       console.log("(+) "+ data);
    }
});
========================================== END ==========================================

========================================== bip.py ==========================================
#!/usr/bin/env python2
import struct
import time
import sys
import argparse
from Crypto.Cipher import AES
from bluepy.btle import Peripheral, DefaultDelegate, ADDR_TYPE_RANDOM

''' TODO
Key should be generated and stored during init
'''

UUID_SVC_MIBAND2 = "0000fee100001000800000805f9b34fb"
UUID_CHAR_AUTH = "00000009-0000-3512-2118-0009af100700"
UUID_SVC_ALERT = "000018020000100080000805f9b34fb"
UUID_CHAR_ALERT = "00002a060000100080000805f9b34fb"
UUID_SVC_HEART_RATE = "0000180d00001000800000805f9b34fb"
UUID_CHAR_HRM_MEASURE = "00002a3700001000800000805f9b34fb"
UUID_CHAR_HRM_CONTROL = "00002a3900001000800000805f9b34fb"

HRM_COMMAND = 0x15
HRM_MODE_SLEEP      = 0x00
HRM_MODE_CONTINUOUS = 0x01
HRM_MODE_ONE_SHOT   = 0x02

CCCD_UUID = 0x2902

class MiBand2(Peripheral):
    _KEY = b'\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x40\x41\x42\x43\x44\x45'
    _send_key_cmd = struct.pack('<18s', b'\x01\x00' + _KEY)
    _send_rnd_cmd = struct.pack('<2s', b'\x02\x00')
    _send_enc_key = struct.pack('<2s', b'\x03\x00')

    def __init__(self, addr):
        Peripheral.__init__(self, addr, addrType=ADDR_TYPE_RANDOM)
        print("Connected")

        svc = self.getServiceByUUID(UUID_SVC_MIBAND2)
        self.char_auth = svc.getCharacteristics(UUID_CHAR_AUTH)[0]
        self.cccd_auth = self.char_auth.getDescriptors(forUUID=CCCD_UUID)[0]

        svc = self.getServiceByUUID(UUID_SVC_ALERT)
        self.char_alert = svc.getCharacteristics(UUID_CHAR_ALERT)[0]

        svc = self.getServiceByUUID(UUID_SVC_HEART_RATE)
        self.char_hrm_ctrl = svc.getCharacteristics(UUID_CHAR_HRM_CONTROL)[0]
        self.char_hrm = svc.getCharacteristics(UUID_CHAR_HRM_MEASURE)[0]
        self.cccd_hrm = self.char_hrm.getDescriptors(forUUID=CCCD_UUID)[0]

        self.timeout = 5.0
        self.state = None
        # Enable auth service notifications on startup
```

```
        self.auth_notif(True)
        self.waitForNotifications(0.1) # Let Mi Band to settle

    def init_after_auth(self):
        self.cccd_hrm.write(b"\x01\x00", True)

    def encrypt(self, message):
        aes = AES.new(self._KEY, AES.MODE_ECB)
        return aes.encrypt(message)

    def auth_notif(self, status):
        if status:
            print("Enabling Auth Service notifications status...")
            self.cccd_auth.write(b"\x01\x00", True)
        elif not status:
            print("Disabling Auth Service notifications status...")
            self.cccd_auth.write(b"\x00\x00", True)
        else:
            print("Something went wrong while changing the Auth Service notifications status...")

    def send_key(self):
        print("Sending Key...")
        self.char_auth.write(self._send_key_cmd)
        self.waitForNotifications(self.timeout)

    def req_rdn(self):
        print("Requesting random number...")
        self.char_auth.write(self._send_rnd_cmd)
        self.waitForNotifications(self.timeout)

    def send_enc_rdn(self, data):
        print("Sending encrypted random number")
        cmd = self._send_enc_key + self.encrypt(data)
        send_cmd = struct.pack('<18s', cmd)
        self.char_auth.write(send_cmd)
        self.waitForNotifications(self.timeout)

    def initialize(self):
        self.setDelegate(AuthenticationDelegate(self))
        self.send_key()

        while True:
            self.waitForNotifications(0.1)
            if self.state == "AUTHENTICATED":
                return True
            elif self.state:
                return False

    def authenticate(self):
        self.setDelegate(AuthenticationDelegate(self))
        self.req_rdn()

        while True:
            self.waitForNotifications(0.1)
            if self.state == "AUTHENTICATED":
                return True
            elif self.state:
                return False

    def hrmStartContinuous(self):
        self.char_hrm_ctrl.write(b'\x15\x01\x01', True)

    def hrmStopContinuous(self):
        self.char_hrm_ctrl.write(b'\x15\x01\x00', True)


class AuthenticationDelegate(DefaultDelegate):

    """This Class inherits DefaultDelegate to handle the authentication process."""
```

```
    def __init__(self, device):
        DefaultDelegate.__init__(self)
        self.device = device

    def handleNotification(self, hnd, data):
        # Debug purposes
        #print("HANDLE: " + str(hex(hnd)))
        #print("DATA: " + str(data.encode("hex")))
        if hnd == self.device.char_auth.getHandle():
            if data[:3] == b'\x10\x01\x01':
                self.device.req_rdn()
            elif data[:3] == b'\x10\x01\x04':
                self.device.state = "ERROR: Key Sending failed"
            elif data[:3] == b'\x10\x02\x01':
                random_nr = data[3:]
                self.device.send_enc_rdn(random_nr)
            elif data[:3] == b'\x10\x02\x04':
                self.device.state = "ERROR: Something wrong when requesting the random number..."
            elif data[:3] == b'\x10\x03\x01':
                print("Authenticated!")
                self.device.state = "AUTHENTICATED"
            elif data[:3] == b'\x10\x03\x04':
                print("Encryption Key Auth Fail, sending new key...")
                self.device.send_key()
            else:
                self.device.state = "ERROR: Auth failed"
            #print("Auth Response: " + str(data.encode("hex")))
        elif hnd == self.device.char_hrm.getHandle():
            rate = struct.unpack('bb', data)[1]
            print("Heart Rate: " + str(rate))
        else:
            print("Unhandled Response " + hex(hnd) + ": " + str(data.encode("hex")))

def main():
    """ main func """
    parser = argparse.ArgumentParser()
    parser.add_argument('host', action='store', help='MAC of BT device')
    parser.add_argument('-t', action='store', type=float, default=3.0,
                        help='duration of each notification')

    parser.add_argument('--init', action='store_true', default=False)
    parser.add_argument('-n', '--notify', action='store_true', default=False)
    parser.add_argument('-hrm', '--heart', action='store_true', default=False)
    arg = parser.parse_args(sys.argv[1:])

    print('Connecting to ' + arg.host)
    band = MiBand2(arg.host)
    band.setSecurityLevel(level="medium")

    if arg.init:
        if band.initialize():
            print("Init OK")
        band.disconnect()
        return
    else:
        band.authenticate()

    band.init_after_auth()

    if arg.notify:
        print("Sending message notification...")
        band.char_alert.write(b'\x01')
        time.sleep(arg.t)
        print("Sending phone notification...")
        band.char_alert.write(b'\x02')
        time.sleep(arg.t)
        print("Turning off notifications...")
        band.char_alert.write(b'\x00')
```

```
    if arg.heart:
        print("Cont. HRM start")
        band.hrmStopContinuous()
        band.hrmStartContinuous()
        for i in range(30):
            band.waitForNotifications(1.0)

    print("Disconnecting...")
    band.disconnect()
    del band


if __name__ == "__main__":
    main()
```

========== END ==========