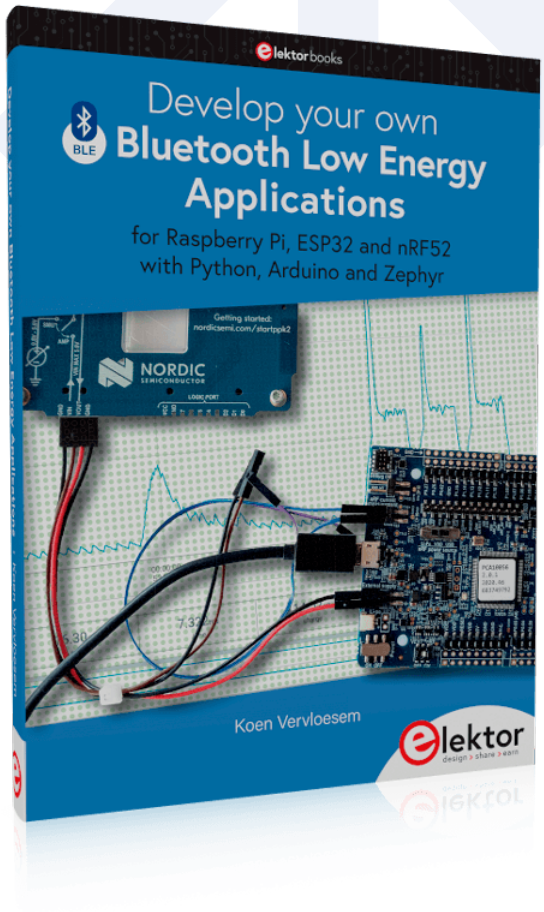
Reverse-engineering van een **Bluetooth Low Energy** LED-badge

een BLE-apparaat besturen met een Python-script

Koen Vervloesem (België)

Veel Bluetooth Low Energy (BLE)-apparaten worden geleverd met hun eigen mobiele app die gebruikerscontrole biedt en doorgaans een aangepast protocol implementeert dat alleen door het apparaat en de bijbehorende app wordt begrepen. En zoals bijna altijd is er geen specificatie die je kunt bestuderen om je eigen implementatie te maken. Het goede nieuws is echter: reverse-engineering van het BLE-apparaat is de manier om het te 'kraken' en met je eigen software te gebruiken. Dat gaat zo.

Opmerking van de redactie: *Hoewel dit artikel gebaseerd is op een hoofdstuk uit het 257 pagina's tellende boek Develop your own Bluetooth Low Energy Applications (Elektor 2022), werd de inhoud door de auteur speciaal voor Elektor Magazine bewerkt tot een compleet en volledig reproduceerbaar project.*



In dit artikel ga ik een BLE LED-badge reverse-engineeren als voorbeeld van deze uitdagende en uiterst leerzame activiteit. Doe mee en ontdek hoe het apparaat werkt door de BLE-services en -kenmerken te onderzoeken. Kijk met me mee hoe ik de bijbehorende mobiele app decompileer en BLE-verkeer tussen de app en het apparaat afluister. Mijn doel is om een Python-script te schrijven om de LED-badge te besturen, zodat ik de officiële mobiele app niet langer nodig heb.

Onderzoek van de LED-badge

De LED-badge van AliExpress zoals afgebeeld in **figuur 1** heeft een 11 x 55 LED-pixeldisplay dat in verschillende kleuren verkrijgbaar is. Het ondersteunt Bluetooth zonder gespecificeerde versie.
Het scannen van de QR-code op de achterkant van de badge resulteert in een "HTTP 404"-fout. Op Google Play vond ik een app met de naam "Bluetooth LED Name Badge", afkomstig van Shenzhen Lesun

Electronics Co. Ltd (**figuur 2**). Deze app kan tekst en pictogrammen naar de badge sturen en biedt een aantal instelmogelijkheden, waaronder het scrolltempo.
Laten we de LED-badge inschakelen en de BLE-scanner-app "nRF Connect for Mobile" [1] gebruiken om te zien wat het apparaat zoal 'adverteert' zoals dat heet. Druk twee keer op de onderste knop. Het display toont dan een BLE-pictogram. In nRF Connect zie je dat het nu fabrikantspecifieke gegevens en twee eigen BLE-services aangeeft: 0xfee7 en 0xfee0 (**figuur 3**). Het apparaat heeft de naam *LSLED*. Door er verbinding mee te maken, worden de services onthuld, samen met de kenmerken die in **tabel 1** zijn opgesomd.

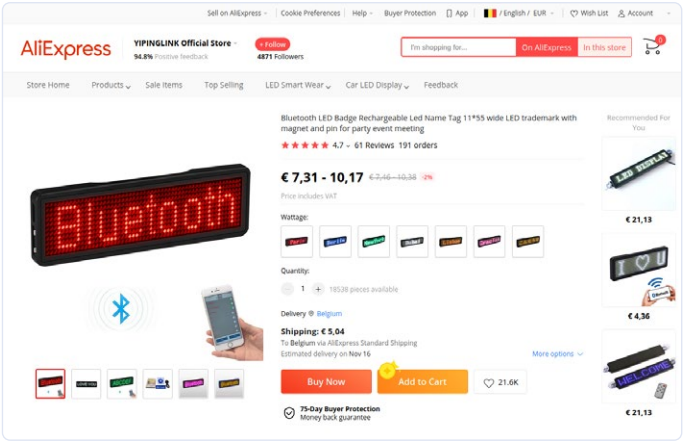
Tabel 1. Karakteristieken van de BLE-badge.

Service	Karakteristiek	Eigenschap
0xfee7	0xfec7	WRITE
0xfee7	0xfec8	INDICATE
0xfee7	0xfec9	READ
0xfee0	0xfee1	NOTIFY, READ, WRITE

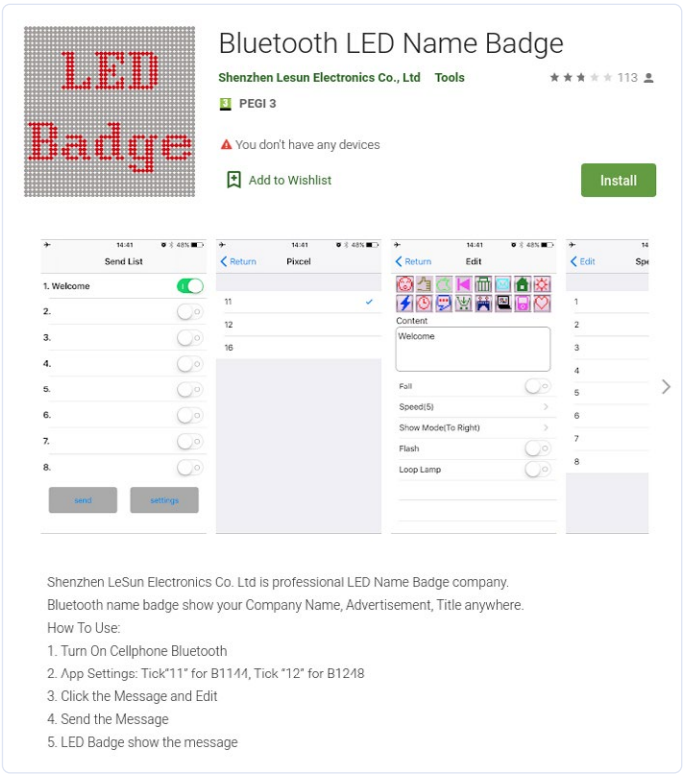
Het lezen van het kenmerk "0xfec9" in nRF Connect for Mobile levert dezelfde waarde op als de fabrikantspecifieke gegevens. Lezen van 0xfee1 retourneert geen gegevens en abonneren op de meldingen levert ook niets op. De Characteristic User Description van 0xfee1 is "Data". Deze methode levert niets bruikbaars op.

De mobiele app decompileren

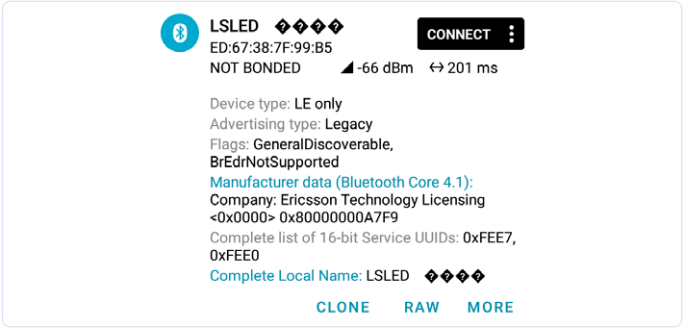
Aangezien duidelijk is dat de Bluetooth LED Name Badge-app voor Android met de LED-badge kan communiceren, zullen we nu eens kijken wat de app eigenlijk doet. We doen dit door de app te decompileren en de werking van de broncode uit te pluizen.
Download het APK-bestand voor de Android-app met behulp van een externe APK-downloadersite zoals [2]. Plak gewoon de Google Play Store-URL [3] van de app in het zoekveld van APKPure. Vervolgens kun je het bestand downloaden. Dit APK-bestand bevat de Dalvik-bytecode die Android kan uitvoeren. Om te begrijpen wat de app doet, hebben we de broncode nodig. Terwijl de ontwikkelaar zijn broncode naar Dalvik heeft gecompileerd, kunnen we dit andersom doen met een decompiler zoals jadx [4], die Dalvik-bytecode omzet in Java-broncode. Met de meest recente release voor Windows [5] kunnen we de grafische interface starten door te dubbelklikken op het **jadx-gui**-bestand in de **bin**-directory. Start op andere besturingssystemen (Linux, MacOS) de grafische interface door bin/jadx-gui uit te voeren vanaf de opdrachtregel.
Open het gedownloade APK-bestand. Het programma decompileert nu de app en toont links een boomstructuur van de packages en Java-bestanden. Nu begint de zoektocht. We hebben al een paar aanknopingspunten: de UUID's van de services en karakteristieken die gevonden worden in nRF Connect for Mobile. In het menu **Navigation / Text search** kun je een paar zoektermen invoeren.



Figuur 1. Deze Bluetooth LED-badge lijkt een interessante kandidaat voor reverse-engineering!



Figuur 2. De "Bluetooth LED Name Badge app" in de Google Play Store kan commando's naar de LED-badge sturen.



Figuur 3. De Bluetooth LED-badge in nRF Connect for Mobile.

Listing 1.

```
public static final String UUID_CHARACTERISTICS_WRITE = "fee1";
public static final String UUID_SERVICE = "fee0";
```

Listing 2.

```
public static byte[] get64(List<SendContent> list, int i) {
     Iterator<SendContent> it = list.iterator();
     while (it.hasNext()) {
         Log.d("abcdef", "get64------------SendContent:" + it.next().toString());
     }
     byte[] bArr = new byte[64];
     bArr[0] = 119;
     bArr[1] = 97;
     bArr[2] = 110;
     bArr[3] = 103;
     bArr[4] = 0;
     bArr[5] = 0;
     bArr[6] = getFlash(list);
     bArr[7] = getMarquee(list);
     byte[] modeAndSpeed = getModeAndSpeed(list);
     for (int i2 = 0; i2 < 8; i2++) {
         bArr[i2 + 8] = modeAndSpeed[i2];
     }
     byte[] msgLength = getMsgLength(list, i);
     for (int i3 = 0; i3 < 16; i3++) {
         bArr[i3 + 16] = msgLength[i3];
     }
     bArr[32] = 0;
     bArr[33] = 0;
     bArr[34] = 0;
     bArr[35] = 0;
     bArr[36] = 0;
     bArr[37] = 0;
     byte[] date = getDate();
     for (int i4 = 0; i4 < 6; i4++) {
         bArr[i4 + 38] = date[i4];
     }
     for (int i5 = 0; i5 < 19; i5++) {
         bArr[i5 + 44] = 0;
     }
     bArr[63] = 0;
     return bArr;
 }
```

Dat is makkelijker gezegd dan gedaan. De code vermeldt de 0xfee7-service of de kenmerken ervan niet! Het vermeldt wel de andere service en zijn kenmerk in de klasse `com.yannis.ledcard.ble.BleDevice`. Relevante delen ervan zijn te zien in **listing 1**.
In de code kun je nu zoeken naar deze twee constanten:

`UUID_CHARACTERISTICS_WRITE`
`UUID_SERVICE`

De eenvoudigste manier om dit te doen is waarschijnlijk door met de rechter muisknop op de naam van de constante te klikken en vervolgens **Find usage** te selecteren. Klik vervolgens op een van de gevonden instanties om het bijbehorende bestand op die plaats te openen.
Tijdens het zoeken naar enkele andere aanwijzingen in de code, kwam ik een veelbelovende codefragment tegen dat afbeeldingen en weergavemodi in de klasse `com.yannis.ledcard.util.LedDataUtil` betreft. Concreet vond ik de Java-methode van **listing 2** die een soort header voor gegevens lijkt te creëren.
We zien hier een vaste header (6 bytes), modus en snelheid, berichtlengte en datum. Hij wordt aangeroepen door een `getSendHeader()` in de genoemde klasse. Het was niet voldoende om uit te vinden wat de app doet door alleen maar verder in deze code te duiken. Dus begon ik de app met de LED-badge te gebruiken terwijl ik het dataverkeer tussen beide apparaten afluisterde. Voor een beter begrip kon ik nu de opgedane kennis uit de broncode van de app combineren met live-dataverkeer.

Afluisteren van het BLE-verkeer tussen badge en app

Wireshark [6] vangt het live Bluetooth-verkeer van je telefoon op met behulp van de Android Debug Bridge. Sluit je telefoon aan op je computer met een USB-kabel en sta de debug-verbinding op je telefoon toe. Start Wireshark. Dit zou **Android Bluetooth Btsnoop Net** moeten tonen als een van de beschikbare interfaces. Na dubbelklikken op de interface, zie je live Bluetooth-verkeer voorbij komen! Een nog betere optie is om de nRF Sniffer voor Bluetooth LE plug-in voor Wireshark [7] te gebruiken, waarbij een nRF52840-dongle wordt gebruikt als het hardwaregedeelte van de 'sniffer.'
Druk op de BLE-badge twee keer op de onderste knop totdat het Bluetooth-pictogram wordt weergegeven. Selecteer vervolgens het apparaat in Wireshark zodat alleen pakketten van en naar dat specifieke apparaat orden weergegeven.
Installeer "Bluetooth LED Name Badge" op je Android-telefoon en open de app. In Wireshark zie je dat je telefoon een scan request doet en de BLE-badge antwoordt met de apparaatnaam in het scan response. Selecteer het apparaattype in de app. Voor de LED-badge van 11 x 55 pixels is dat type 11. Tik op **yes**.

De app toont een lijst met te verzenden berichten. De standaardconfiguratie is om één bericht te verzenden: **Welcome**. Tik op **Send**. De app maakt nu verbinding met de LED-badge en toont het welkomstbericht op het display (**figuur 4**).
In Wireshark zien we een CONNECT_REQ-pakket gevolgd door een verzoek om attributen. Dan komen er een paar Write Requests en reacties. Je krijgt een beter overzicht door met de rechter muisknop op de Write Request-opcode in het Bluetooth Attribute Protocol-gedeelte van de packet-details te klikken en vervolgens **Apply as Filter / Selected** te kiezen. Je kunt ook het volgende invoeren: `btatt.opcode == 0x12` als displayfilter.
In dit geval heeft de app negen Write Request-packets verzonden, elk met 16 bytes aan gegevens, zoals hieronder weergegeven:

```
77 61 6e 67 00 00 00 00 30 30 30 30 30 30 30 30
00 07 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 e5 0a 18 0c 37 25 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 c6 c6 c6 c6 d6 fe ee c6 82 00 00 00 00 00 7c
c6 fe c0 c6 7c 00 00 38 18 18 18 18 18 18 18 3c
00 00 00 00 00 7c c6 c0 c0 c6 7c 00 00 00 00 00
7c c6 c6 c6 c6 7c 00 00 00 00 00 ec fe d6 d6 d6
c6 00 00 00 00 00 7c c6 fe c0 c6 7c 00 00 00 00
```

Als je de Java-methode `get64()` uit de vorige paragraaf bekijkt, herken je de delen van de header: 77 61 6e 67 (hexadecimaal equivalent van decimaal 119 97 110 103). Dan krijgen we twee keer 00 en dan weer twee keer 00, wat de waarden zijn voor respectievelijk `getFlash()` en `getMarquee()`. De volgende acht bytes zijn allemaal 0x30; deze staan voor de modus en snelheid. De volgende 16 bytes zijn de berichtlengte. Omdat de app een lijst van acht berichten toont, zou dit de lijst met de lengtes van deze berichten kunnen zijn. De eerste twee bytes hier zijn 00 07, wat inderdaad het aantal tekens is in de tekst **Welcome**. De veronderstelling is dat de volgende 14 bytes de lengte vertegenwoordigen van de volgende zeven berichten (in dit geval geen).
Dan zijn er zes nullen; de volgende zes bytes vertegenwoordigen de datum, in dit geval e5 0a 18 0c 37 25. Omgerekend naar decimaal is dat 229 10 24 12 55 37. Ik heb deze app uitgevoerd op 24 oktober 2021 om 12:55:37. De maand, dag en tijd klopten. De header eindigt met 20 stuks 00.
Hierna vertegenwoordigen vijf pakketten van 16 bytes op de een of andere manier de zeven tekens van de **Welcome-**tekst. Laten we eens kijken hoe het werkt, te beginnen met een eenvoudiger bericht. Open de app opnieuw en klik op het eerste bericht. Wijzig de tekst in **W** en schakel **Flash** in . Keer terug naar het hoofdscherm, klik op het tweede bericht en voeg deze tekst toe: **e**. Stel voor dit tweede bericht de snelheid in op 8, de modus op **right** en schakel **Marquee** in. Keer terug naar het hoofdscherm. Schakel vervolgens de schuifregelaar naast het tweede bericht in en klik op **Send** terwijl je de Bluetooth Attribute Protocol-packets in Wireshark in het oog houdt.
De LED-badge toont nu de letter W die naar links beweegt en aan en uit knippert. Hierna toont het de letter e die met dubbele snelheid naar rechts beweegt en een frame van bewegende pixels eromheen.
In dit geval stuurde de app zes Write Request-packets, elk met 16 bytes aan data – zie hieronder:

Figuur 4. De Bluetooth LED-badge toont een welkomstbericht.

```
77 61 6e 67 00 00 01 02 30 71 30 30 30 30 30 30
00 01 00 01 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 e5 0a 18 10 06 13 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 c6 c6 c6 c6 d6 fe ee c6 82 00 00 00 00 00 7c
c6 fe c0 c6 7c 00 00 00 00 00 00 00 00 00 00 00
```

Het eerste packet begint met dezelfde 6 bytes, maar dan is de waarde van `getFlash()` 01 en die van `getMarquee()` 02. Dit komt doordat we **Flash** hebben ingeschakeld voor het eerste bericht en **Marquee** voor het tweede bericht.
De volgende byte is nog steeds 0x30, omdat we niets hebben veranderd aan de modus en snelheid van het eerste bericht. Maar de byte hierna luidt nu 0x71. We hebben de snelheid gewijzigd in 8 en de modus in Right. Dus de linker nibble van die byte codeert blijkbaar de snelheid (snelheid 4 is gecodeerd als 3 en snelheid 8 als 7) en de rechter nibble de modus. Het tweede packet toont 00 01 als de lengte van het eerste bericht en hetzelfde voor de lengte van het tweede bericht. Dit bevestigt onze veronderstelling. Dan is de datum gecodeerd.
Er zijn nu nog twee packets over. Converteer de hexadecimale waarden naar binair en formatteer ze byte voor byte en op volgorde, en splits ze vervolgens in elf rijen (ja, want het scherm is 11 pixels hoog), zoals hieronder:

```
00000000
11000110
11000110
11000110
11000110
11010110
11111110
11101110
11000110
10000010
00000000

00000000
00000000
00000000
00000000
01111100
11000110
11111110
11000000
11000110
01111100
00000000
```

Listing 3.

```
"""Find BLE LED badges.
Copyright (c) 2022 Koen Vervloesem
SPDX-License-Identifier: MIT
"""
import asyncio
from bleak import BleakScanner
num_devices = 0
def device_found(device, advertisement_data):
    """Show device details if it's a BLE LED badge."""
    global num_devices
    if device.name.startswith("LSLED"):
        num_devices += 1
        print(
            f" () - RSSI: "
        )
async def main():
    """Scan for BLE devices."""
    print("Searching for LED badges...")
    scanner = BleakScanner()
    scanner.register_detection_callback(device_found)
    await scanner.start()
    await asyncio.sleep(5.0)
    await scanner.stop()
    if not num_devices:
        print("No devices found")
if __name__ == "__main__":
    asyncio.run(main())
```

Tegen het einde worden ter opvulling ('filler') nullen gebruikt om een blok van 16 bytes op te vullen (hier niet getoond). Als je dezelfde decodering toepast op het eerste bericht met **Welcome** en vervolgens de bitmaps van de letters naast elkaar legt, zie je een 1 voor elke ingeschakelde pixel en een 0 voor elke uitgeschakelde, zoals hier:

```
00000000000000000000000000000000000000000000000000000000000000
11000110000000000011100000000000000000000000000000000000000000
11000110000000000011000000000000000000000000000000000000000000
11000110000000000011000000000000000000000000000000000000000000
11000110011111000001100001111100011111000111011000111110000
11010110110001100001100011000110110001101111111011000110
11111110111111100001100011000000110001101101011011111110
11101110110000000001100011000000110001101101011011000000
11000110110001100001100011000110110001101101011011000110
10000010011111000011110001111100011111001100011001111100
00000000000000000000000000000000000000000000000000000000
```

We hebben nu een vrij goed idee van het formaat van de gegevens die naar de LED-badge moeten worden gestuurd om tekst op het display weer te geven. Laten we dit nu allemaal combineren en een Python-script schrijven om eigen afbeeldingen naar de LED-badge te sturen.

Willekeurige afbeeldingen naar de LED-badge schrijven met Bleak

Met de Bluetooth LED-naambadge kun je ook enkele voorgedefinieerde kleine 11 x 11-bitmaps verzenden. Maar nu je weet in welk formaat je data moet verzenden, kun je bijvoorbeeld ook een willekeurige bitmap sturen die het 11 x 55-display helemaal vult. Laten we dit doen met behulp van Bleak [8], een platformonafhankelijke Python-bibliotheek voor BLE. Tik in:

```
pip3 install bleak
```

Laten we eerst een script maken dat gedurende vijf seconden naar BLE-badges scant op basis van hun naam en vervolgens hun Bluetooth-adressen en RSSI's toont – zie **listing 3**.
Als je dit uitvoert met twee LED-badges die beide in de luister-modus staan, toont het script:

```
$ python3 find_led_badge.py
Searching for LED badges...
ED:67:38:80:0D:E2 (LSLED) - RSSI: -74
ED:67:38:7F:99:B5 (LSLED) - RSSI: -83
```

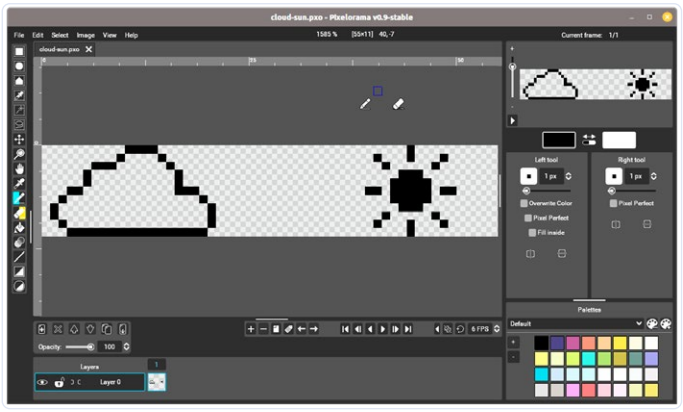
Laten we nu voortbouwen op de kennis die we in de vorige paragraaf hebben opgedaan. Bekijk dit script dat de juiste commando's naar de LED-badge schrijft om een afbeelding weer te geven – zie **listing 4**.
Naast Bleak gebruikt dit script ook de Pillow-bibliotheek [9]:

```
pip3 install Pillow
```

De eerste vier BLE-commando's zijn hardgecodeerd. In het eerste commando definieert 0x34 de modus en snelheid voor het eerste bericht. De '4' staat voor statische modus: het bericht wordt weergegeven als een stilstaand beeld. In de tweede opdracht definieert 0x07 de lengte van het bericht als het aantal 8-pixel-tekens. Het displey is 55 pixels breed en 7 keer 8 is gelijk aan 56, dus we moeten de afbeelding coderen als 7 tekens. De volgende twee commando's zouden de huidige datum moeten coderen, maar dat is niet echt nodig. Stuur gewoon 16 nullen voor elk van deze commando's.
`chunks()` is een hulpfunctie om een byte-array te splitsen in een lijst met byte-arrays van elk 16 bytes. Dit wordt gebruikt in de functie `commands_for_image()` die een afbeeldingsbestand converteert naar bytes die tekens op het scherm vertegenwoordigen. Aan het einde van de functie worden deze bytes opgesplitst in blokken van 16 bytes. `commands_for_image()` opent een afbeelding met Pillow en laadt de pixels. Voor elk van de zeven tekens op het scherm codeert de functie het teken als 11 bytes: één voor elke rij. Al deze bytes worden toegevoegd aan `image_bytes`.
Aan het einde van de afbeelding wordt de byte-array opgevuld met extra bytes, zodat het een veelvoud is van 16 bytes. En uiteindelijk wordt het opgesplitst in hapklare stukken van 16 bytes.
De functie `main()` veegt alles bij elkaar. Hij maakt verbinding met het apparaat en maakt een lijst met commando's: de vier commando's van de header worden uitgebreid met de commando's voor de afbeelding. Vervolgens wordt elk van deze opdrachten naar het kenmerk geschreven met UUID 0xfee1. De hoofdcode onderaan controleert of je twee

Listing 4.

```
"""Display an image on a BLE LED badge.
Copyright (c) 2022 Koen Vervloesem
SPDX-License-Identifier: MIT
"""
import asyncio
import sys
from PIL import Image
import bleak
WRITE_CHAR_UUID = "0000fee1-0000-1000-8000-00805f9b34fb"
COMMAND1 = bytes([0x77, 0x61, 0x6E, 0x67, 0x00, 0x00, 0x00, 0x00, 0x34, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30])
COMMAND2 = bytes([0x00, 0x07, *(14 * [0x00])])
COMMAND3 = bytes(16 * [0x00])
COMMAND4 = bytes(16 * [0x00])
def chunks(lst, n):
    """Yield successive n-sized chunks from lst."""
    for i in range(0, len(lst), n):
        yield lst[i : i + n]
def commands_for_image(image):
    """Return commands to show an image on the BLE LED badge."""
    image_bytes = bytearray()
    with Image.open(image) as im:
        px = im.load()
        for i in range(7):  # 7x8 = 56 -> 7 bytes next to each other
            for row in range(11):
                row_byte = 0
                for column in range(8):
                    try:
                        pixel = int(
                            px[(i * 8) + column, row][3] / 255
                        )
                    except IndexError:
                        pass  # Ignore the 56th pixel in a full row
                    row_byte = row_byte | (pixel << (7 - column))
                image_bytes.append(row_byte)
    # Fill the end with zeroes to have a multiple of 16 bytes
    image_bytes.extend(bytes(16 - len(image_bytes) % 16))
    # Split image bytes into 16-byte chunks
    return list(chunks(image_bytes, 16))
async def main(address, filename):
    """Connect to BLE LED badge and send commands to show an image."""
    try:
        async with bleak.BleakClient(address) as client:
            commands = [COMMAND1, COMMAND2, COMMAND3, COMMAND4]
            commands.extend(commands_for_image(filename))
            for command in commands:
                await client.write_gatt_char(WRITE_CHAR_UUID, command)
    except asyncio.exceptions.TimeoutError:
        print(f"Can't connect to device .")
    except bleak.exc.BleakError as e:
        print(f"Can't write to device : ")
if __name__ == "__main__":
    if len(sys.argv) == 3:
        address = sys.argv[1]
        filename = sys.argv[2]
        asyncio.run(main(address, filename))
    else:
        print(
            "Please specify the BLE MAC address and image filename."
        )
```



Figuur 5. Pixelorama is een open-source 2D-afbeeldingseditor.

Figuur 6. Door de juiste BLE-commando's te verzenden, kun je opscheppen met je eigen afbeeldingen van 11 x 55 pixels op de LED-badge.

argumenten op de opdrachtregel hebt opgegeven. De eerste is toegewezen aan het Bluetooth-adres en de tweede aan de bestandsnaam. Voordat je deze code uitvoert, moet je een afbeelding maken van exact 11 x 55 pixels, met behulp van een pixeleditor zoals Pixelorama [10]. Vul een pixel voor elke LED die je op het display wilt laten oplichten. **Figuur 5** geeft een voorbeeld van een afbeelding van een wolk en zon die ik in Pixelorama heb gemaakt.
Exporteer de afbeelding als een .png-bestand. Zet nu de LED-badge in Discovery-modus door twee keer op de knop onderaan te drukken, totdat het Bluetooth-pictogram verschijnt. Voer het Python-script uit met twee argumenten; eerst het Bluetooth-adres van de badge en vervolgens de bestandsnaam van de afbeelding:

```
$ python3 display_led_badge.py ED:67:38:7F:99:B5 cloud-sun.png
```

Tot slot

De LED-badge zou nu 'jouw' afbeelding moeten weergeven (**figuur 6**). Het is dus gelukt: je hebt met succes de BLE-badge 'gekraakt' en kunt deze nu met je eigen code gebruiken. Het script dat ik hier heb gepresenteerd is zeker voor verbetering vatbaar, vooral om het algemener toegankelijk te maken. Ik hoop dat het je inspireert om soortgelijke reverse engineering toe te passen op andere BLE-apparaten.

220439-03

Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail maar de auteur via koen@vervloesem.eu of naar de redactie van Elektor via redactie@elektor.com.

WEB LINKS

[1] nRF Connect for Mobile-app: https://www.nordicsemi.com/Products/Development-tools/nrf-connect-for-mobile
[2] APK-bestand : https://apkpure.com
[3] De app in Google Play Store: https://play.google.com/store/apps/details?id=com.yannis.ledcard
[4] De jadx-decompiler : https://github.com/skylot/jadx
[5] Laatste release: https://github.com/skylot/jadx/releases
[6] Wireshark: https://www.wireshark.org
[7] nRF Sniffer for Bluetooth LE : https://www.nordicsemi.com/Products/Development-tools/nrf-sniffer-for-bluetooth-le
[8] Bleak : https://bleak.readthedocs.io
[9] De Pillow-bibliotheek: https://pillow.readthedocs.io
[10] Pixelorama: https://orama-interactive.itch.io/pixelorama
[11] Info en resources bij het boek: https://www.elektor.com/develop-your-own-bluetooth-low-energy-applications
[12] Code op GitHub: https://github.com/koenvervloesem/bluetooth-low-energy-applications