Projekt Zespołowy

Sczytywanie danych z inteligentnej opaski

Autorzy:
Mikołaj Skubisz
Szymon Witusiak

Prowadzący:
dr inż. Łukasz Jeleń K9/W4

Termin zajęć:
ŚR 13:00-16:00

2020

1 Wstęp

1.1 Cel projektu

Celem projektu było stworzenie aplikacji mobilnej na system Android umożliwiającej połączenie oraz komunikację z inteligentną opaską FitGo FW 11 Light, poprzez standard Bluetooth Low Energy (LE). Urządzenie to wykonuje pomiary parametrów biologicznych użytkownika, m.in.: puls, ciśnienie, poziom tlenu we krwi, ilość kroków wykonanych w danym dniu; z tego powodu za główną funkcjonalność aplikacji uznano możliwość wywoływania pomiaru oraz pobranie wyniku z opaski, po czym przedstawienie go użytkownikowi w przystępnej formie.

Istotną przeszkodą w realizacji głównego celu projektu jest fakt, że opaska jest przystosowana jedynie do współpracy z aplikacją stworzoną przez producenta urządzenia (LinkSelf), jednocześnie będąc zabezpieczoną przed pobieraniem danych przez obce aplikacje. W celu rozwiązania tego problemu koniecznym było wykorzystanie dekompilatorów, konwertujących plik z rozszerzeniem .apk, do postaci plików .smali lub .java, przez co możliwy jest wgląd w domniemany kod programu, gdzie ukryte są zamknięte techniki używane przez twórców do interakcji z badanym urządzeniem bluetooth. Jako że pliki .java są zdecydowanie najbardziej przystępne do odczytu przez programistę, zdecydowano się na dekompilacje aplikacji do tego formatu.

Po rozszyfrowaniu odpowiednich informacji, pomyślnie stworzono mechanizm pobierający dane sensoryczne, przez co dostępne są one z poziomu aplikacji. Na chwilę obecną dane pomiarowe (puls, poziom tlenu, ciśnienie krwi) prezentowane użytkownikowi wyświeltane są przy pomocy prostego GUI stworzonego w Android Stuido (przy czym dodatkowe dane w postaci ciągu bajtów dostępne są dla programisty w postaci logów w owym IDE).

Rysunek 1: Opaska użyta w projekcie - FitGo FW11 Light

1.2 Wymaganie funkcjonalne

- Po uruchomieniu aplikacji sprawdzane jest w pierwszej kolejności czy Bluetooth jest obsługiwany przez urządzenie.
- Gdy urządzenie obsługuje Bluetooth sprawdzany jest jego stan. Jeśli nie jest on włączony, uruchamiany jest on od razu po włączeniu aplikacji, ponad to wysuwana jest prośba o udostępnienie lokalizacji w celu detekcji urządzeń BLE
- Jako widok startowy uruchamiany jest panel gdzie użytkownik może wybrać informację jaką chcę wyświetlić (np. puls, ciśnienie) oraz co istotne uruchomić ustawienia w których inicjuje połączenie z opaską (połączenie Bluetooth LE).
- Po nawiązaniu połączenia użytkownik ma możliwość wysłać żądanie do opaski w celu inicjacji pomiarów
- Po poprawnie wysłanym żądaniu dane są pobierane z urządzenia. W celu wyświetlenia odpowiedniej informacji, należy wybrać odpowiedni przycisk/ikonę - by wynieść na pierwszy plan nową aktywność związaną z odpowiednią daną.
- Użytkownik może uzyskać informacje o: poziomie tlenu w krwi, tętnie, ciśnieniu krwi, ilości kroków/dzień
- Do prezentacji konkretnego wyniku z sensorów zastosowane zostało proste GUI.

1.3 Wymagania niefunkcjonalne

- Aplikacja stworzona będzie na urządzenia mobilne z systemem Android (minimum Android 7.0)
- Urządzenie musi wspierać Bluetooth 4.0
- Jako że w projekcie zachodzi interakcja z sensorami, do komunikacji między urządzeniami wykorzystany zostanie Bluetooth Low Energy
- Do stwirzenia aplikacji użyte zostanie IDE - Android Studio
- Do interakcji z opaską wykorzystano oficjalne API, stworzone przez inżynierów Android'a
- Język programowania użyty do napisania aplikacji - JAVA
- Użyty system kontroli wersji - GIT.

2 Omówienie technologii wykorzystanych przy tworzeniu aplikacji

2.1 Bluetooth Low Energy

Bluetooth Low Energy (Bluetooth 4.0) jest to standard komunikacji bezprzewodowej którego priorytetem jest energooszczędność podczas transmisji danych. Z tego powodu ma on zastosowanie przy wymianie danych z urządzeniami, których słabością jest zużycie energii. Przykładami takich urządzeń są różnego rodzaju czujniki (np. medyczne), czujniki zbliżeniowe czy wszelkiego rodzaju urządzenia IoT. Jako że badana opaska wpisuje się perfekcyjnie w ramy wymienionych wyżej urządzeń, wspiera ona BLE, przez co cała aplikacja oparta jest właśnie na tym standardzie.

Cechą charakterystyczną połączenia z urządzeniem BLE jest fakt że przed tą akcją nie dokonywana jest operacja parowania jak w przypadku klasycznego Bluetooth. Spowodowane jest to tym, że podczas parowania urządzeń przeprowadzane jest szyfrowanie, które wymaga pewnej mocy obliczeniowej, której niektóre urządzenia Bluetooth Low Energy nie posiadają.

Istotnym do zrozumienia zastosowanych technik jest zapoznanie się z kluczowymi koncepcjami technologii Bluetooth LE.

2.1.1 GATT/ATT

ATT (Attribute Protocol) jest to protokół wykorzystywany przez urządzenia BLE, definiujący w jaki sposób dane na urządzeniu przedstawiane są klientowi. Jego podstawowym budulcem są atrybuty - małe kawałki danych wystawione przez serwer, o ściśle zdefiniowanej postaci (uchwyt, UUID (określający typ atrybutu), opis).

GATT natomiast wykorzystuje ATT - jest to mechanizm grupujący, hierarchizujący atrybuty w serwisy, charakterystyki. W skrócie tworzy on z atrybutów większą całość, umożliwiającą spójne przedstawianie danych z urządzenia LE.

2.1.2 Characteristic

Charakterystyka, jest to część profilu GATT przechowująca wartość np. w naszym przypadku wynik przeprowadzonego pomiaru. Poza wartością zawiera również informacje o poziomie dostępu do niej - charakterystyki mogą mieć dostęp m.in.: READ, WRITE, NOTIFY, przy czym pojedyncza charakterystyka może zawierać wiele pozwoleń. Określona jest przez adres UUID.

2.1.3 Servis

W najbardziej intuicyjnej, a jednocześnie wystarczajej do zrozumienia działania aplikacji postaci, serwis (określony przez adres UUID) jest to zbiór charakterystyk. Serwis grupuje charakterystyki w logiczny, powiązany tematycznie zbiór. Przykładowo serwis dotyczący pomiaru temperatury może zawierać 3 charakterystyki: charakterystykę zawierającą wartość w stopniach Celsjusza, charakterystykę zawierającą wartość pomiaru w stopniach Fahrenheita, charakterystykę przechowującą timestamp pomiaru.

2.1.4 Descriptor

W uproszczeniu deskryptor jest to dodatkowa wartość skojarzona z konkretną charakterystyką, używana do opisu wartości na niej. Przykładowo mając charakterystykę zawierającą pomiar temperatury, przy pomocy deskryptora możemy określić że dany pomiar wykonany został w jednostce Celsius

2.1.5 GATT Client/ GATT Server

W standardzie Bluetooth Low Energy, profil GATT defniuje sposób w jaki dwa urządzenia kooperują ze sobą. Jedno z urządzeń przyjmuje rolę klienta (GATT Client) - wysyła żądania do serwera w celu otrzymania odpowiedzi (np. wyników pomiaru), drugie zaś serwera (GATT Server) - odpowiada ono na żądania, np. udostępniając wartości pobrane z czujników. W przedstawianym projekcie opaska jest serwerem, tworzona aplikacja natomiast przyjmuje rolę klienta.

2.2 Bytecode Viewer (Zestaw dekompilatorów)

W celu znalezienia metody komunikacji oryginalnej aplikacji z opaską oraz uzyskania adresów UUID serwisów oraz charakterystyk odpowiedzialnych za przedstawianie wartości pomiaru na urządzeniu, wykorzystano oprogramowanie open sourcowe Bytecode Viewer - https://github.com/Konloch/bytecode-viewer. Jest to aplikacja służąca do odwrotnej inżynierii. zawierająca wiele użytecznych narzędzi, z czego najistotniejszym dla stworzenia aplikacji był zestaw dekompilatorów, dzięki któremu otrzymano przybliżony kod aplikacji, przez co poznano techniki w niej zastosowane. Istotną przewagą tego multinarzędzia nad pojedyńczymi dekompilatorami, jest to że nie każda aplikacja do dekompilacji działa poprawnie dla całości kodu, dlatego też oprogramowanie to zwalnia z obowiązku pobierania osobno wielu dekompilatorów, przez co zdecydowanie ułatwia pracę w obszarze reverse engineering.

2.3 Wireshark (Bluetooth traffic)

Po zastosowanieu Bytecode Viewera, dekompilacja aplikacji umożliwiła poznanie sposobu w jaki oryginalna aplikacja komunikuje się z opaską, natomiast same żądania - ciągi bajtów wysyłane jako komendy okazały się być niejasne (spowodowane jest to swego rodzaju próbą ukrycia komend przez twórców, poprzez uzależnienie ich od licznych flag ustawianych w kodzie). Jako że analizowanie wszystkich możliwości wartości żądań nie kalkulowało się czasowo, zdecydowano się przejrzeć ruch Bluetooth na urządzeniu (smartfonie z zainstalowaną aplikacją). Plik z logami Bluetooth traffic uzyskano poprzez uruchomienie w ustawieniach telefonu opcji Bluetooth HCI snoop log. Następnie w pliku z rozszerzeniem .cfa zapisywane są bajty wchodzące oraz wychodzące z urządzenia poprzez protokół Bluetooth. W celu otworzenia i zapoznania się z tymi danymi, wykorzystano aplikację Wireshark, która jest analizatorem pakietów, umożliwiającym przeglądanie zebranych logów, w plikach o interesującym nas rozszerzeniu. Skutkiem tych działań było poprawne przeanalizowanie oraz rozszyfrowanie interesujących nas ciągów bajtów.

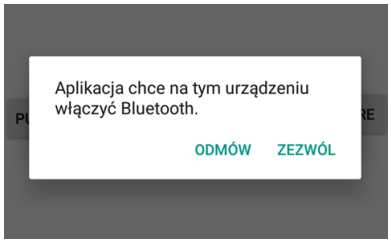
3 Implementacja oraz testy aplikacji

3.1 Wykrywanie urządzeń oraz połączenie z opaską

3.1.1 Uruchomienie funkcji bluetooth

Pierwszym założeniem aplikacji było uruchomienie funkcji bluetooth natychmiastowo po starcie aplikacji, w celu umożliwienia dalszych operacji. Punkt ten zrealizownao przy pomocy obiektu BluetoothAdapter (uprzednio w pliku AndroidManifest.xml nadając odpowiednie zezwolenia), dzięki któremu istnieje możliwość interakcji z radiem bluetooth urządzenia.

Ponad to, użytkownik jest proszony o zezwolenie na użycie lokalizacji telefonu, co wymagane jest podczas łączenia z urządzeniami Low Energy. Zarówno prośba o uruchomienie bluetooth jak i o udostępnienie lokalizacji realizowana jest poprzez wyskakujące okienko, gdzie osoba korzystająca z aplikacji musi wyrazić zgodę na odpowiadające żądania aplikacji.



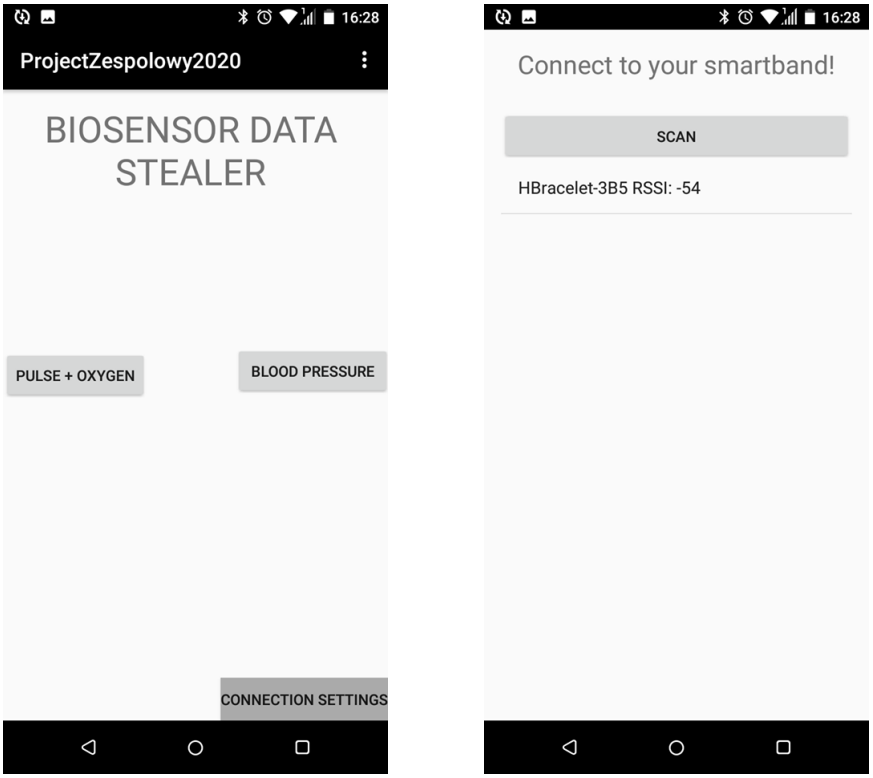
Rysunek 2: Okno umożliwiające zezwolenie Bluetooth

3.1.2 Wykrywanie urządzeń

Po udzieleniu odpowiednich zgód, użytkownik znajduje się w głównym menu aplikacji (za które odpowiada aktywność - MainActivity). Pierwszą istotną rzeczą którą należy zrobić w celu uzyskania charakterystyk z opaski, jest wykrycie urządzenia BLE oraz nawiązanie połączenia. W tym celu użytkownik musi przejść do okna ustawień - przy pomocy przycisku - CONNECTION SETTINGS, gdzie uruchamiana jest nowa aktywność obsługująca m.in. skanowanie oraz połączenie z urządzeniami.

Po ukazaniu się widoku ustawień, trzymając opaskę w pobliżu telefonu, należy wcisnąć przycisk SCAN, który inicjuje proces wykrywania urządzeń LE. W tym momencie, wywoływanie jest funkcja z API odpowiadająca za szukanie urządzeń, a jednocześnie z 10 sekundowym opóźnieniem umieszczana jest w kolejce komunikatów dla głównego wątku metoda przerywająca detekcję. Owy przedział czasowy zdeterminowany jest faktem, iż proces szukania urządzeń jest energochłonny, przez co istotnym jest ograniczenie czasu skanowania (po nieudanej próbie, istnieje możliwość ponownej inicjacji całego procesu).

W przypadku wykrycia danego urządzenia LE, przez aplikację, wynik zwracany jest w skojarzonym callback'u, gdzie następnie sprawdzono moc odbieranego sygnału. Empirycznie podjęto decyzję o dodawaniu do zbioru wykrytych urządzeń tylko te, których RSSI, jest większe od -70, w celu ograniczenia analizowanych urządzeń do minimum. Gdy odpowiednie urządzenie zostanie wykryte, dodawane jest ono do struktury HashSet, aby uniknąć powtórzeń wystąpienia tego samego urządzenia, wśród zbioru wykrytych.



Rysunek 3: Okno główne oraz okno ustawień aplikacji

3.1.3 Połączenie z opaską

Wyniki wyszukiwań urządzeń wyświetlane są na liście w oknie ustawień aplikacji. W celu nawiązania połączenia z konkretnym, należy wybrać odpowiednią pozycję z możliwych. W tym momencie, jeżeli nazwa urządzenia jest zgodna z badaną opaską (ciąg liter zaczyna się od "HBracelet"), uruchamiany jest w tle główny serwis odpowiedzialny za połączenie oraz komunikację z urządzeniem.

W pierwszej kolejności po inicjacji serwisu, następuję wywołanie metody, nawiązującej połączenie z opaską. Wyniki tej operacji otrzymywane są poprzez skojarzony callback, który obsługuje zmianę stanu połączenia z urządzeniem, jak i również wykrywanie oraz zmianę wartości charakterystyk oraz wykrywanie serwisów. W przypadku zmiany stanu połączenia, gdy podłączenie do opaski zakończy się sukcesem, użytkownik informowany jest o tym odpowiednim toast'em, a następnie uruchamiane jest wykrywanie wszystkich serwisów dostępnych w ramach urządzenia. Ponad to, zdarzenie to do możliwego obsłużenia wysyłane jest poprzez metodę sendBroadcast, do odpowiedniego, słuchającego BroadcastReceivera, ulokowanego w aktywności związanej z wykrywaniem urządzeń. Po wyżej wymienionych operacjach urządzenie jest gotowe do czytania charakterystyk.

3.2 Czytanie dostępnych charakterystyk

Po nawiązaniu połączenia, detekcji serwisów oraz poznaniu ich UUID, aplikacja jest w stanie czytać dostępne charakterystyki na urządzeniu. Głównym celem tego procesu było znalezienie charakterystyk odpowiedzialnych za wyniki pomiarów z urządzenia. Po wielokrotnym wyświetlaniu charakterystyk, zaobserwowano iż wartości w nich przechowywane są zakodowane (jedyną, którą udało się rozszyfrować to nazwa urządzenia), ale co istotniejsze, niezmienne, z tego powodu przestały być one przedmiotem dalszych zainteresowań. Wyjątkiem natomiast jest jedna charakterystyka zawierająca bazowo wartość null, na której możliwe jest ustawienie powiadomień, w przypadku zmian wartości. Uruchomiło to jak się później okazało słuszne podejrzenia do tego, że jest to miejsce gdzie w czasie rzeczywistym umieszczane są wyniki pomiarów.

W celu weryfikacji słuszności tego założenia, zastosowano dekompilator JADX-Core, w ramach programu Bytecode Viewer. Na podstawie analizy zdekompilowanej aplikacji ustalono że: urządzenie posiada 2 istotne charakterystyki w ramach tego samego serwisu - jedna z prawami do wpisywania wartości, druga zaś z prawami do odczytu, oraz ustawiania powiadomień w przypadku zmian na niej. Na podstawie mechanizmów użytych w systemie, wywnioskowano iż są to swego rodzaju 2 kanały komunikacyjne - jeden do wysyłania żądań (przykładowo o pomiar ciśnienia w czasie rzeczywistym), natomiast drugi do odbierania wyników z urządzenia pomiarowego. Co ciekawe, UUID owych charakterystyk oraz serwisu zakodowane zostało sztywno w kodzie, przez co ustalenie tych wartości odbyło się bezproblemowo, jednocześnie pozytywnie weryfikując wcześniejsze osądy co do jedynej charakterystyki z możliwością ustawienia powiadomień.

```
pub  public class GattLayer implements OnConnectListener, OnOperationListener, TryTimeStrategy {
    p     private static final boolean D = true;
    /     /* access modifiers changed from: private */
    p     public static final UUID NAME_CHARACTERISTIC_UUID = UUID.fromString("0000ff04-0000-1000-8000-00805f9b34fb");
    /     /* access modifiers changed from: private */
    p     public static final UUID NOTIFY_CHARACTERISTIC_UUID = UUID.fromString("0000ff03-0000-1000-8000-00805f9b34fb");
    /     /* access modifiers changed from: private */
    p     public static final UUID SERVICE_UUID = UUID.fromString("000001ff-3c17-d293-8e48-14fe2e4da212");
    p     private static final String TAG = "GattLayer";
    /     /* access modifiers changed from: private */
    p     public static final UUID WRITE_CHARACTERISTIC_UUID = UUID.fromString("0000ff02-0000-1000-8000-00805f9b34fb");
    /     /* access modifiers changed from: private */
```

Rysunek 4: Adresy UUID serwisu oraz istotnych charakterystyk, w zdekompilowanej aplikacji.

3.3 Ustawianie powiadomień na głównej charakterystyce.

Po ustaleniu charakterystyki, w której przechowywane są wartości pomiarów, kluczowym było ustawienie notyfikacji w przypadku zmieniającej się wartości. Osiągnięto to poprzez metodę, zezwalającą urządzeniu Android na otrzymywanie powiadomień. Jednakże funkcja ta nie włącza powiadomień po stronie urządzenia pomiarowego (opaski), przez co koniecznym było dodatkowe wpisanie wartości uruchamiającej notyfikację, do deksryptora skojarzonego z charakterystyką, odpowiadającego za żądaną przez nas operację.

3.4 Proces deszyfracji wysyłanych ciągów bajtów

Kolejnym krokiem, po ustaleniu odpowiednich adresów UUID, było znalezienie komend (ciągu bajtów wysyłanych do opaski), inicjujących oraz kończących dany pomiar na urządzeniu. By to osiągnąć po raz kolejny użyto mechanizmu dekompilacji. Główną strategią uzyskania pożądanych informacji było "przejście po wywoływanych metodach" od fragmentu związanego przykładowo z pomiarem ciśnienia do momentu wpisywania wartości do charakterystyki odpowiadającej za nasłuchiwanie komend, gdzie najistotniejszym był argument przekazywany do metody - writeCharacteristic().

Po analizie zauważono zastosowany mechanizm komplikujący rozszyfrowanie pożądanych informacji. Ciąg bajtów wysyłanych, nie jest zakodowany na sztywno, tak jak przykładowo adresy UUID, natomiast jest on obliczany na podstawie wielu zmiennych zależnych od pewnych stanów, zdarzeń w systemie aplikacji. Z tego powodu zbiór potencjalnych odpowiedzi na zadany problem drastycznie powiększa się. Ze względu na czasochłonny proces uzyskiwania wszystkich możliwych ciągów bajtów oraz weryfikacje odpowiedzi metodą prób i błędów, zdecydowano się porzucić podejście uzyskiwania komend poprzez analizę zdekompilowanego kodu aplikacji. Wyniki tego podejścia nie były jednak całkowicie do odrzucenia, gdyż dzięki tej metodzie udało się ustalić że komenda wywołująca pomiar jest tablicą bajtów o długości 17, kończąca się wartościami: [... , 0x04, 0x00, 0x01, 0x05, 0x02].

```
private static byte[] preparePacket(byte[] bArr, boolean z, boolean z2, int i, int i2) {
    int i3;
    int i4;
    if (bArr != null) {
        i4 = bArr.length;
        i3 = bd_crc16(bArr, i4);
    } else {
        i4 = 0;
        i3 = 0;
    }
    byte[] bArr2 = new byte[8];
    byte[] bArr3 = new byte[(i4 + 8)];
    bArr2[0] = MAGIC_BYTE;
    bArr2[1] = (byte) ((z2 ? z ? 48 : 16 : 0) | (i & 15));
    bArr2[2] = (byte) ((i4 >> 8) & 255);
    bArr2[3] = (byte) (i4 & 255);
    bArr2[4] = (byte) ((i3 >> 8) & 255);
    bArr2[5] = (byte) (i3 & 255);
    bArr2[6] = (byte) ((i2 >> 8) & 255);
    bArr2[7] = (byte) (i2 & 255);
    System.arraycopy(bArr2, 0, bArr3, 0, 8);
    if (i4 > 0) {
        System.arraycopy(bArr, 0, bArr3, 8, i4);
    }
    return bArr3;
}
```

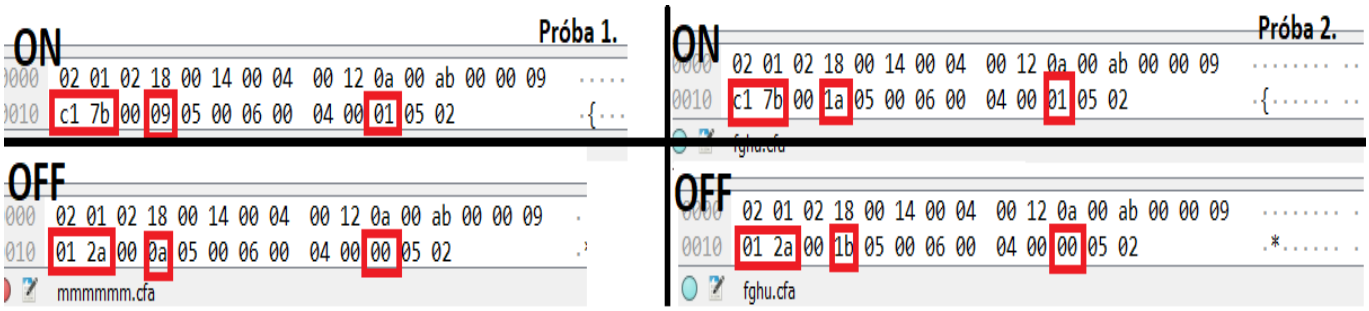
Rysunek 5: Fragment aplikacji tworzący żądanie wysyłane do opaski

Podejściem na które zdecydowano się po dekompilacji pliku .apk, była analiza ruchu bluetooth, na podstawie pliku .cfa, zawierającego logi, opisujące wysyłane oraz otrzymywane pakiety bluetooth na urządzeniu z systemem Android. By były one wpisywane w plik o podanym rozszerzeniu należy uruchomić opcję w ustawieniach programistycznych na telefonie, zezwalającą przechwytywanie logów HCI na telefonie.

Po przeniesieniu pliku na komputer (przy pomocy np. Google Drive), aplikacją która umożliwia

jego czytanie jest program Wireshark. Po przeprowadzonej analizie pakietów wychodzących, namierzono wiadomości o długości zgodnej z długością wywnioskowaną na podstawie dekompilacji aplikacji - 17 bajtów, jednocześnie zwracając uwagę na ostatnie bity w danym żądaniu. Dzięki temu uzyskano tablicę wpisywaną do charakterystyki WRITE, w celu uzyskania pomiarów pulsu, poziomu tlenu oraz ciśnienia.

Istotnym podkreślenia jest fakt, że przy każdorazowym wysłaniu komendy, jej wartość nie jest identyczna - 8 bajt w kolejności "od lewej" zmienia się. Po przeprowadzeniu testów, liczba ta nie ma znaczącego wpływu na zwracaną odpowiedź przez opaskę, natomiast zauważyć można że ciąg wysyłany w celu zaprzestania pomiaru jest prawie identyczny jak rozpoczynający je (Rysunek 6.). Trzema istotnymi różnicami są: wartość 5 i 6 komórki zostaje zmieniona, wyzerowany zostaje trzeci bajt "od końca", wartość w 8 kolumnie podniesiona zostaje o 1 względem tej wartości w ciągu wcześniej wysłanym. Z powodu braku znajomości zachowania opaski na wartości z poza schematu wyczytanego z logów, zdecydowano się nie modyfikować rozwiązań zastosowanych w oryginalnej aplikacji, przez co nie sprawdzono czy inkrementacja liczby znajdującej się w odpowiedniej komórce jest niezbędna do zaprzestania pomiaru.



Rysunek 6: Screen z roboczej analizy wysyłanych pakietów.

3.5 Uzyskiwanie danych pomiarowych z opaski

Jako że próba uzyskania informacji o wartości tablic wysyłanych jako żądania do rozpoczęcia/zakończenia pomiaru zakończyła się sukcesem, w tworzonej aplikacji zaimplementowano rozwiązania umożliwiające czytanie wyników pomiaru pulsu, poziomu tlenu we krwi oraz ciśnienia skurczowego, rozkurczowego.

W tym celu stworzono dwie nowe aktywności - jedną związaną z pomiarem pulsu oraz tlenu, drugą z pomiarem ciśnienia. Przy pomocy przycisków użytkownik jest w stanie wywołać pomiar. W tym momencie ustawiana jest w systemie flaga oznaczająca trwające badanie - w celu uniknięcia kilkukrotnego wysłania tego samego polecenia do opaski. Następnie ustawiany jest na charakterystyce, oczekującej na komendy, ciąg bajtów uzyskany w procesie opisanym w poprzednim punkcie, przy czym 8 bajtowi nadano wartość 0x64. Z tego powodu tablica bajtów rozpoczynająca badanie w naszej aplikacji, przedstawia się następująco (w zapisie heksadecymalnym): [0xab, 0x00, 0x00, 0x09, 0xc1, 0x7b, 0x00, 0x64, 0x05, 0x00, 0x06, 0x00, 0x04, 0x00, 0x01, 0x05, 0x02].

Po wysyłaniu żądania opaska początkowo wysyła informacje zwrotną (którą jak wszystkie inne odczytujemy z charakterystyki NOTIFY), oznaczającą prawdopodobnie powiadomienie o rozpoczęciu badania. Następnie regularnie co kilka sekund na charakterystyce umieszczane są pary tablic - pierwsza sygnalizująca nierozszyfrowaną do teraz informację (być może informowanie o typie nad-

chodzącej wartości), druga z pożądanym pomiarem.

Po otrzymaniu tablicy, w której zakodowane zostały wyniki badań, kluczowym było określenie formatu, oraz miejsca gdzie znajdują się wyniki odpowiednich wartości (np. pulsu). Dokonano tego na podstawie obserwacji - wykonywano pomiary, po czym patrzono, które wielkości przychodzące ulegają zmianie. Zauważono, że wartości z pomiarów zapisywane są w możliwie najprostszej formie (UINT8), przez co bez większych problemów określono na których miejscach w tablicy znajduje się wynik badań: pulsu (pozycja 11 licząc od 0), tlenu (pozycja 12 licząc od 0), ciśnienia skurczowego (pozycja 14 licząc od 0), ciśnienia rozkurczowego (pozycja 13 licząc od 0). Ciągi bajtów przychodzące po wysłaniu komendy z aktywności PulseActivity/BloodPressureActivity wyświetlane są m.in. w logach Android Studio (Rysunek 7.)

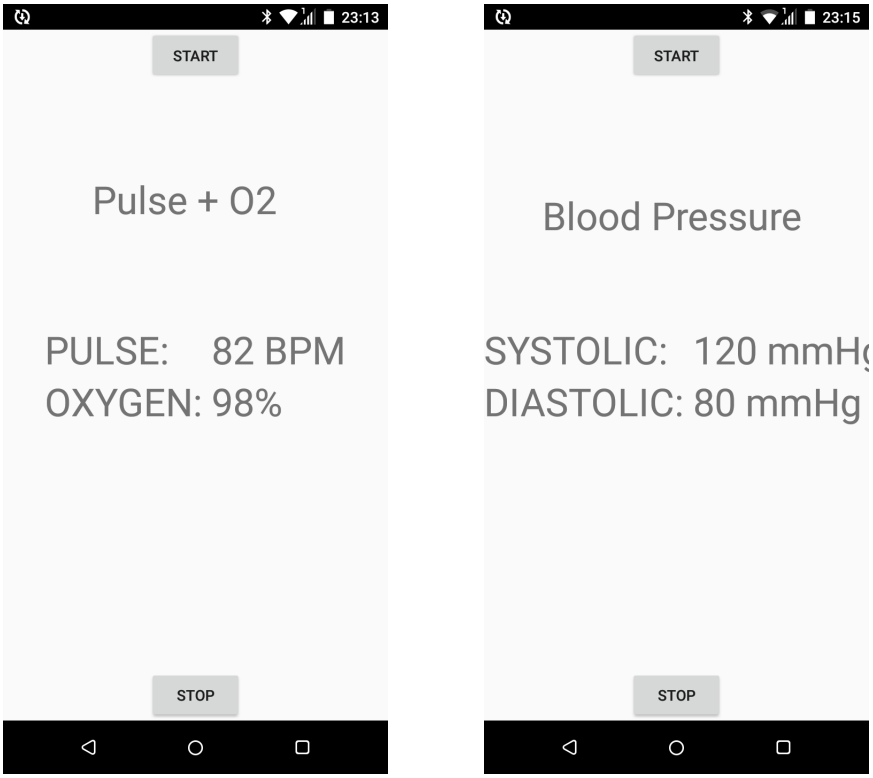
```
W/RenderThread: type=1400 audit(0.0:29472): avc: denied { read } for name="boot_mode" dev="sy
I/ConnectionService: VALUE: [-85, 16, 0, 0, 0, 0, 0, 64]
I/ConnectionService: VALUE: [-85, 0, 0, 16, 46, 83, 0, 0]
I/ConnectionService: VALUE: [5, 0, 21, 0, 11, 40, -65, 0, 1, 5, 72, 88, 96, 81, 122, 16]
I/PulseActivity: Pulse: 88
    Oxygen: 96
D/OpenGLRenderer: CacheTexture 3 upload: x, y, width height = 0, 0, 476, 506
W/RenderThread: type=1400 audit(0.0:29473): avc: denied { read } for name="boot_mode" dev="sy
I/ConnectionService: VALUE: [-85, 0, 0, 16, 2, 86, 0, 1]
I/ConnectionService: VALUE: [5, 0, 21, 0, 11, 40, -65, 0, 1, 5, 72, 87, 99, 80, 120, 12]
I/PulseActivity: Pulse: 87
    Oxygen: 99
D/OpenGLRenderer: CacheTexture 3 upload: x, y, width height = 344, 93, 38, 54
```

Rysunek 7: Wartości zwracane po wywołaniu pomiaru z aktywności PulseActivity.

3.6 Implementacja GUI

Jako że po wykonaniu wszystkich poprzednich akcji, jesteśmy w stanie kontrolować przeprowadzany pomiar, jednocześnie otrzymywać informacje o jego wynikach, ostatecznym krokiem było zaimplementowanie GUI, gdzie informacje będą przedstawiane już nie tylko programiście, lecz także użytkownikowi. W tym celu w menu głównym utworzono 2 przyciski, przenoszące do odpowiednich aktywności związanych z pulsem i tlenem oraz z ciśnieniem krwi. Po wybraniu odpowiedniego przycisku, otwierane jest nowe okno, gdzie umieszczone są 2 przyciski - rozpoczynające oraz kończące pomiar, oraz 2 pola tekstowe odpowiedzialne za prezentowanie wyniku pomiaru (odpowiednio pulsu i tlenu oraz ciśnienia skurczowego i rozkurczowego) - odświeżające swoją zawartość każdorazowo po otrzymaniu informacji o nowej próbce z badania (Rysunek. 8).

Po ukończeniu tego etapu, uznano że wersja podstawowa tworzonej aplikacji została osiągnięta.



Rysunek 8: Okno aktywności PulseActivity oraz BloodPressureActivity.

4 Podsumowanie

Po przebrnięciu przez etapy opisane w powyższych punktach, można bezapelacyjnie stwierdzić, iż projekt stworzenia aplikacji sczytującej pomiary z bio-czujników zakończył się sukcesem. Analizując wykonaną pracę, widać że najbardziej czasochłonnym, sprawiającym najwięcej problemów etapem była analiza wysyłanych i otrzymywanych pakietów. Powodem tego jest przede wszystkim zamknięte oprogramowanie oryginalnej aplikacji, przez co konieczne było zastosowanie dekompilatorów. Dodatkowo wartości wysyłane na opaskę nie są łatwo dostępne w kodzie, dlatego też ich odczyt wymagał poświęcenia czasu.

Ponad to wartym uwagi jest fakt, że opaska poza przeanalizowanymi w projekcie pomiarami przeprowadzonymi w czasie rzeczywistym, oferuje badania m.in. ilości kroków, czy też spalonych kalorii w danym dniu. Z tego powodu projekt w przyszłości można rozszerzyć o przejmowanie również tych pomiarów. Na tę chwilę projekt w wersji podstawowej jest ukończony.

5 Bibliografia

- Opis działania Bluetooth Low Energy: https://epxx.co/artigos/bluetooth_gatt.html
- Omówienie Bluetooth LE w kontekście tworzenia aplikacji na system Android: https://developer.android.com/guide/topics/connectivity/bluetooth-le
- Bytecode Viewer: https://github.com/Konloch/bytecode-viewer/
- Program Wireshark: https://www.wireshark.org/
- Opis procesu analizy ruchu bluetooth: https://support.honeywellaidc.com/s/article/How-to-capture-Bluetooth-traffic-from-and-to-an-Android-Device