Универзитет у Београду
Електротехнички факултет


# Сигурност Bluetooth механизама за откривање периферија рачунарских система

Дипломски рад

Ментор:
Др Павле Вулетић, професор

Кандидат:
Јована Вучковић 2012/0086

Београд, Септембар 2024.

# САДРЖАЈ

# 1. Увод

У првим верзијама модерних персонализованих рачунара коришћена је уграђена тастатура као једини уређај за кориснички унос. Убрзо следи развој и коришћење независних уређаја који се по потреби могу докупљивати и додавати на рачунар помоћу каблова - периферије. Како је број доступних периферија порастао, тако је и кориснички радни сто постао загушен са кабловима који повезују исти. Да би се корисник ослободио каблова, и добио могућност произвољног смештања и коришћења ових периферија, а да не буде везан дужином каблова, развијен је протокол *Bluetooth*. Коришћење овог протокола је дало опције за унапређење коришћења и намене употребе мобилних телефона у генерацију паметних телефона, са опцијом за повезивање на системе бежичних слушалица, повезивање са штампачима и брзу штампу слика, и слично.

*Bluetooth* протокол пружа изузетну брзину за пренос и репродукцију звука, као и пренос датотека и других података између упарених (два) уређаја. Надоградња протокола у нови, *Bluetooth Low Energy* протокол се јавља ради још једноставнијег проналаска упаривих периферијских уређаја, и лаку претрагу и излиставање доступних и познатих (некада упарених) уређаја у простору. Увођењем и коришћењем BLE протокола појављује се комуникација између уређаја помоћу огласних пакета, који не захтева да су уређаји у упареном стању. Помоћу огласних пакета, сви уређаји у радном простору могу знати да је оглашени уређај такође у том простору, и колико далеко се налази од њих. Коришћење података о мобилном уређају који је у близини, и закључавању рачунара када се тај исти мобилни уређај одаљи од рачунара, су база за сигурностну функционалност *Dynamic Lock*. Да би смо поуздано користили ову функционалност и све податке које добијамо кроз огласне пакете морамо бити сигурни да су они послати и добијени од ауторизованих уређаја, и да је интегритет тих пакета сачуван од пошиљаоца до примаоца. Са преносивим уређајима за рад, било да је у питању персонализовани рачунар, или паметни телефон, важно нам је да је заштита тих уређаја поуздана, и да протокол комуникације и произвођачи (као и корисници) развијају ову технологију са сигурношћу на уму.

У поглављу 2 је објашњен протокол *Bluetooth*, разлог развоја протокола *Bluetooth Low energy*, као и преглед начина успостављања комуникације и безбедност истих. У другој половини поглавља је дискутована функционалност *Dynamic Lock* као пример сигурносног пропуста и првим верзијама ове функционалности.

Поглавље 3 чини опис радног окружења коришћеног за симулацију напада, опис тока напада, и напослетку анализа кључних делова кода којим је напад извршен.

# 2. BLUETOOTH ПРОТОКОЛ И DYNAMIC LOCK

Поглавље даје основе *Bluetooth BT* и *Bluetooth Low Energy BLE* протокола са анализом процеса оглашавања, препознавања, и упаривања уређаја. У другом делу поглавља је представљен *Dynamic lock* , као нова функционалности оперативног система *Windows* која користи BLE протокол да би одредила удаљеност поуздане периферије од рашунара на коме се налази, и да би на основу те удаљености по потреби закључала рачунар.

## 2.1. Bluetooth протокол

*Bluetooth* је протокол бежичне комуникације на фреквенцијама кратког домета (*short-range Radio Frequency RF*). Користи се за креирање бежичних личних мрежа (*Wireless Personal area networks - WPANs*) у циљу упаривања различитих типова периферија за репродукцију и унос података, са радним системом.[6] Одлика система је *point-to-point P2P* комуникација, која мора бити успостављена пре него што може почети размена података између та два уређаја.[9]

Рад на *Bluetooth* технологији, под овим именом, почиње унутар компаније *Ericcson током године 1994. Bluetooth Special Interest Group (SIG)* се формира као непрофитна група компанија тада заинтересованих за стандардизацију и коришћење ове технологије: *Ericsson, IMB, Intel, Nokia*, и *Toshiba*. Прва стандардизована верзија протокола се појављује 2002. године.[6]

## 2.2. Bluetooth Low energy протокол

Као одговор на потражњу “лаких” уређаја за комуникацију, који су више пасивни и не захтевају непрестану комуникацију, конекцију, и самим тим, користе мање енергије, изграђен је протокол за нископотрошне уређаје – *Bluetooth Low Energy (BLE).* Овај протокол се појављује први пут уз *Bluetooth* 4.0 (који је објављен јуна 2010.), из кога вуче корен[6][10]. Поред опција на употребу протокола без потребе за P2P активном комуникацијом, BLE уводи нове технологије за праћење извора сигнала и одређивање не само удаљености, већ и позиције уређаја (слика 2.2.1).

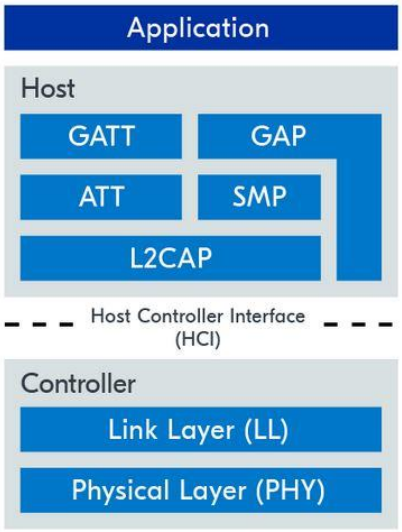
Периферни уређаји (попут лампица, мерача рада срца, различитих типова сензора) више не захтевају непрекидну везу са уређајем који га контролише, већ ће уређај, на основу огласних пакета те периферије, имати увид у статус уређаја, и само по потреби креирати конекцију. Уређаји могу ослушкивати много више уређаја, док су пре BLE протокола морали бити повезани на сваки од њих да би пратили стања, или користити додатан контролни уређај.

| | Bluetooth Low Energy (LE) | Bluetooth Classic |
|---|---|---|
| Frequency Band | 2.4GHz ISM Band (2.402 – 2.480 GHz Utilized) | 2.4GHz ISM Band (2.402 – 2.480 GHz Utilized) |
| Channels | 40 channels with 2 MHz spacing<br>(3 advertising channels/37 data channels) | 79 channels with 1 MHz spacing |
| Channel Usage | Frequency-Hopping Spread Spectrum (FHSS) | Frequency-Hopping Spread Spectrum (FHSS) |
| Modulation | GFSK | GFSK, π/4 DQPSK, 8DPSK |
| Data Rate | LE 2M PHY: 2 Mb/s<br>LE 1M PHY: 1 Mb/s<br>LE Coded PHY (S=2): 500 Kb/s<br>LE Coded PHY (S=8): 125 Kb/s | EDR PHY (8DPSK): 3 Mb/s<br>EDR PHY (π/4 DQPSK): 2 Mb/s<br>BR PHY (GFSK): 1 Mb/s |
| Tx Power* | ≤ 100 mW (+20 dBm) | ≤ 100 mW (+20 dBm) |
| Rx Sensitivity | LE 2M PHY: ≤-70 dBm<br>LE 1M PHY: ≤-70 dBm<br>LE Coded PHY (S=2): ≤-75 dBm<br>LE Coded PHY (S=8): ≤-82 dBm | ≤-70 dBm |
| Data Transports | Asynchronous Connection-oriented<br>Isochronous Connection-oriented<br>Asynchronous Connectionless<br>Synchronous Connectionless<br>Isochronous Connectionless | Asynchronous Connection-oriented<br>Synchronous Connection-oriented |
| Communication Topologies | Point-to-Point (including piconet)<br>Broadcast<br>Mesh | Point-to-Point (including piconet) |
| Positioning Features | Presence: Advertising<br>Direction: Direction Finding (AoA/AoD)<br>Distance: RSSI, Channel Sounding | None |

**Слика 2.2.1 Упоредни преглед спецификација и опција BLE и BT протокола[9]**

Кључни делови API BLE протокола[6], приказани на слици 2.2.2:

- GATT – *Generic ATTribute Profile*, процедуре за рад са атрибутима (ATT)
- GAP – *Generic Access Profile* је слој који комуницира са апликацијом и пружа подршку за откривање и повезивање уређаја
- ATT – *ATTribute Protocol* енкапсулација и одржавање података уређаја
- SMP – *Security Manager Protocol* за сигурносну комуникацију
- L2CAP – Logical Link Control and adaptation Protocol




**Слика 2.2.2. Логички приказ протокола[2]**

## 2.3. Улоге уређаја у комуникацији

Протокол подржава два модела рада – први, у коме BLE активно или пасивно тражи конекцију са другим BLE уређајем, али напослетку жели конекцију, или BLE уређаји који су само одашиљачи, без опција да се са њима неко повеже.

### *2.3.1. Централа*

Ово је тип уређаја који активно скенира и активно иницира везу са другим уређајима. Централа може (не би требала, али није јој ни забрањено) да се оглашава као присутна у простору. Пошто јој је улога скенера и иницијатора, углавном се оглашава са пасивним *beacon*-ом, без имена уређаја. Пошто нема име, већина програма корисника је неће пријављивати као доступну, тј видљиву, али ће неки други уређај из припадајућег система моћи да види да је централа доступна.

### *2.3.2. Периферија*

Улога овог типа уређаја је да се оглашава да је доступан и спреман за конекцију. Он никада не иницира конекцију, нити врши скенирање, иако му то није забрањено (скенирање се код периферија користи у Bluetooth mash-у који није део овог рада).

### *2.3.3. Посматрач*

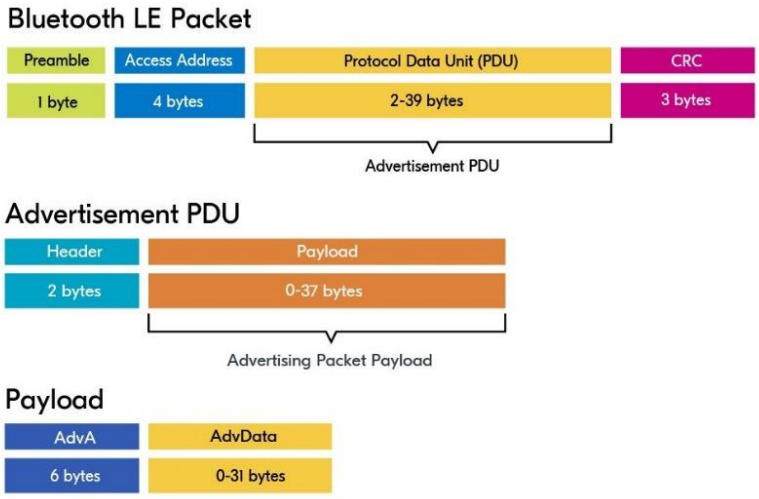
За разлику од централе, посматрач не успостављају везу са другим уређајем, нити се оглашава, већ само чита, прикупља, и анализира пакете у систему.

### *2.3.4. Одашиљач*

Чисто оглашивачки уређај који, слично периферији, шаље пакете, али у тим пакетима назначава да се ца њим не може успоставити веза, или ако то није постављено, у обавези је да игнорише било који пакет упућен ка процесу успостављања везе.

## 2.4. Пакет оглашавања уређаја

Структура пакета оглашавања се налази на слици 2.4




**Слика 2.4. Делови пакета оглашавања[2]**

У иницијалном огласном пакету, налазиће се адреса AdvA која не мора да буде права физичка адреса уређаја, већ је у улози привремене адресе, и AdvData који садржи садржи листу оглашавајућих података (покривени званичном спецификацијом) као што су `BT_DATA_NAME_COMPLETE, BT_DATA_URI, Service UUID, BT_DATA_MANUFACTURER_DATA.`

Ако је скенер уређај заинтересован за детаљнију листу функционалности и детаља, може послати захтев оглашивачу.

## 2.5. Сигурност протокола BT и процес упаривања

Bluetooth стандард прописује 5 безбедносних модела рада уређаја, и пар уређаја мора комуницирати по истом моделу током једне сесије. Један уређај физички може комуницирати са уређајем А у једном безбедносном моду рада, а касније са уређајем Б у другом - модел рада није физичко ограничење уређаја, већ физичка одлика и договор пара уређаја. Модели рада и нивои на којима се ти безбедносни модели примењују се налазе у табели 2.5.

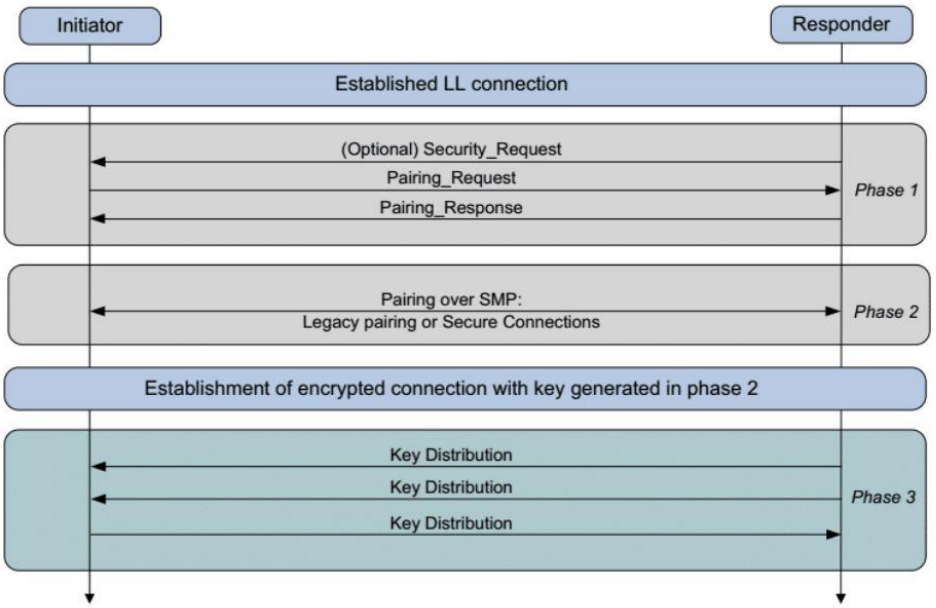
| Модел | Опис | Сигурносне процедуре се врше током креирања |
|---|---|---|
| 4 | Захтева се употреба аутентификованог кључа и захтева се енкриптовање пакета (безбедна веза) | Сервиса и везе (пре и после успостављања физичке везе) |
| 3 | Захтева се употреба се аутентификованог кључа | Везе (пре успостављања физичке везе) |
| 2 | Користи се неаутентификовани кључ | Сервиса (након успостављања физичке везе) |
| 1 | Сигурносни механизми се не користе | / |
| 0 | Без сигурнсоних механизама - Модел дозвољен само за SDP | / |

**Табела 2.5. Модели/нивои безбедности**

Стандард прописује безбедносне смернице за[6]:

- Аутентификацију - идентитет уређаја у комуникацији се врши на основу адресе.
- Поверљивост - спречавање прислушкивања на подацима који се размењују између уређаја
- Ауторизацију - обезбеђује да контролу над неким ресурсом (другим уређајем) може добити само уређај коме је пре контакта то одобрено
- Интегритет размењених порука - поруке и пакети послати од једног уређаја ка другом нису промењени током преноса
- Упаривање уређаја - креира се бар један, или више, парова кључева којима се може верификовати партнер у комуникацији.

Упаривање (*Pairing, Bounding*) се користи да би два уређаја разменили сигурносне кључеве, и тиме започеле сигурну енкриптовану комуникацију. Ово се врши у три фазе, не укључујући иницијално препознавање:



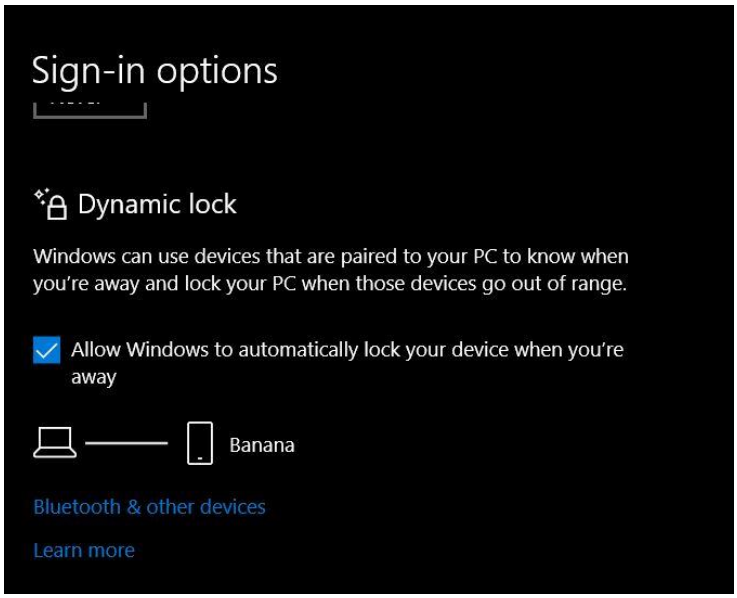

**Слика 2.5.3. Трофазни процес сигурне комуникације при упаривању[1]**

- Иницијатор упаривања шаље захтев за упаривање другом уређају, и добија одговор. Ако је позитиван, иде се у фазу два. Ова фаза не мора да буде сигурносна.
- Упаривање са привременим кључем (може бити размењен директо између уређаја, видети слику 2.1, овде би уобичајно на упаривању рачунара и паметног телефона на не-иницијатору изашао сигурносни број, а на иницијатору опција да се потврди тај сигурносни број), овде почиње сигурносна комуникација.
- Овде се размењују кључеви који ће бити коришћени током било које будуће конекције или откривања ова два уређаја докле год бар један од њих не

избрише податак о упаривању, тачније кључеве. Ако се брисање деси, процес упаривања се понавља, и нови кључеви креирају.

## 2.6. Dynamic Lock функционалност на Windows оперативним системима

*Dynamic Lock* је функционалност која аутоматски закључава *Windows* уређај када сигнал претходно упареног bluetooth уређаја падне испод преодређено конфигурисане максималне вредности за *Recieved Signal Strength Indicator (RSSI).* Ова функција отежава некоме да физички приступи рачунару особе која се одаљила од рачунара и заборавила да га закључа[8].



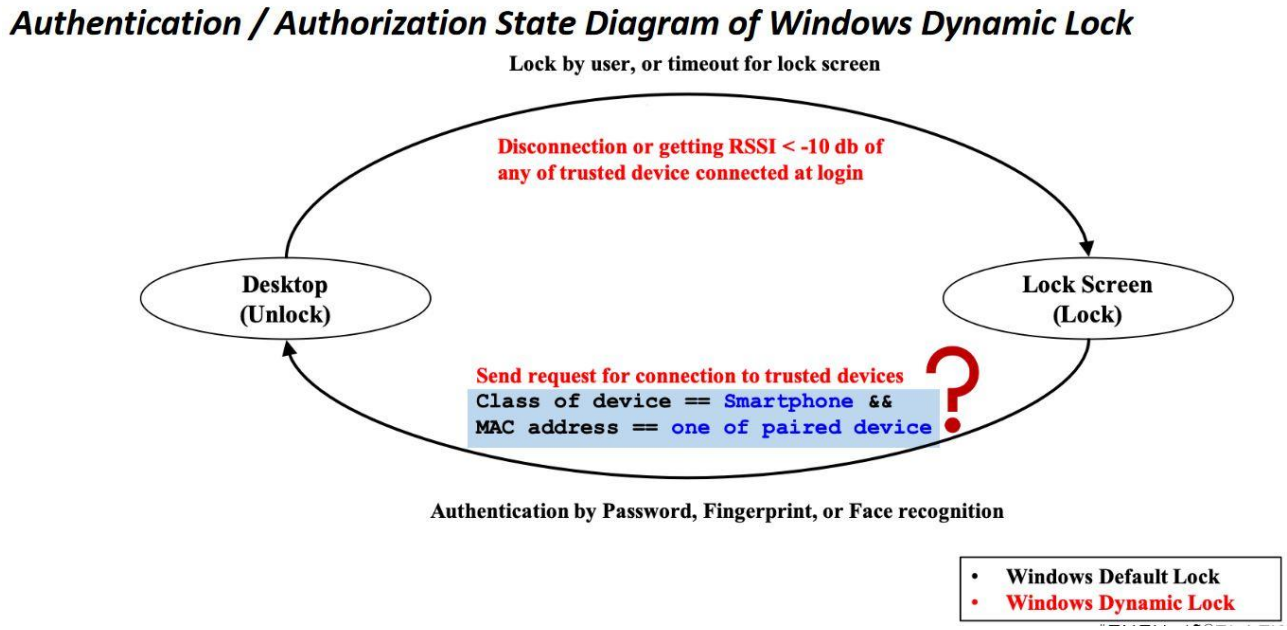

**Слика 2.6. Изглед и локација функционалности *Dynamic lock***

Закључавање се врши када је уређај претходно упарен са рачунаром био присутан у последњем откључавању или покретању рачунара, и када је сигнал уређаја довољно слаб да би рачунар закључио да је уређај далеко.

Први пут се појављује у верзији *Windows 10 Creator's update* (15031), пакет за фебруар 2017.[7].

## 2.7. Начин рада Dynamic Lock-а

У иницијалној верзји *Dynamic Lock*-а, упрошћено говорећи, рачунар је проверавао да ли је *Bluetooth* уређај паметни телефон, и да ли му је *MAC* адреса већ позната. При оваквом начину рада претходно стање упарености се користило као процес додавања *MAC* адресе у индекс поузданих уређаја.




**Слика 2.7.1. Дијаграм стања *Dynamic Lock* функционалности[3]**

На слици 2.7.1 се налази дијаграм логичких стања током рада функционалности Dynamic lock, која закључава рачунар уз помоћ рачунања близине мобилног телефона као периферног сигурног уређаја. Прати се сценарио почевши од стања када је рачунар откључан, до тренутка закључавања.

a) Рачунар је у једном тренутку био упарен са уређајем типа паметног телефона, описано у поглављу 2.4. По извршењу процеса упаривања рачунар препознаје овај уређај као сигуран уређај. Идентификација паметног телефона као сигурног уређаја се врши тако што ће рачунар послати уређају *beacon* са упитом о атрибутима, а уређај ће му одговорити са пакетом који садржи праву *MAC* адресу уређаја (видети b)). На овај начин рачунар и уређај нису у обавези да буду упарени да би имали информацију о томе колико су удаљени један од другог.

b) Уређај ће на упитни *beacon* послати своје праве податке (слика 2.2.2, компонента *Generic Access Profile - GAP* – приметити да овај слој може самостално да повезује апликативни и хардверски ниво, или може да, заједно са *Security Manager Protocol (SMP)* компонентом направи сигурносну конекцију). Током иницијалне фазе оглашавања уређај није у обавези да пружи своју праву MAC адресу.

c) Изглед пакета који је одговор на упитни *beacon* рачунара, а који шаље мобилни уређај (креиран у GAP-у) се налази на слици 2.7.2. У овом тренутку је важан сигурносни модул у коме ови уређаји комуницирају, тачније, у ком сигурносном модулу су договорили комуникацију током упаривања. У упитном *beacon*-у се може наћи захтев за податке заштићене произвољним сигурносним модулом, на који ће уређај одговорити (2.7.2 *Insecure* и *Secure*, сигурносни део садржи све податке који су и у несигурном делу). Ако модул

није захтеван (на пример, неки пре неупарен уређај шаље захтев), увек се пошаље одговор са модулом сигурности 0 (2.7.2 *Insecure*).

d) Рачунар ће, на основу примљених података који су одговор на упитни *beacon*, упоредити *MAC* адресу, и израчунати удаљеност помоћу добијеног *Recieved Signal Strength Indicator* (*RSSI)* податка. По званичном објашњењу[8], подесива вредност (кроз конфигурационе фајлове оперативног система) *rssiMin (-10dB)* је минимална вредност која је потребна да се уређај сматра да је у разумној близини рачунара. Још једна вредност, такође конфигурабилна, је *rssiMaxDelta (-10dB)* је растојање након којег (-20) се сматра да је уређај далеко и креће се у закључавање рачунара. Анализа ових прорачуна, као и одступање износа резултата овог пројекта од званичних упутстава *Dynamic Lock* функционалности није тема овог рада, и неће се даље дискутовати.

У уоченој проблематици са првом верзијом *Dynamic lock*-а није експлицитно тражено да уређај користи енкриптоване податке и да чита заштићене податке пакета у тачки d), и самим тим, било који уређај може одрадити све ове кораке, снимити оба *beacon* пакета, и опонашати уређај, или чак и модификовати вредности (нпр претварати се да је уређај ближи него што јесте, да уређај више није телефон, и слично) у несигурном делу пакета.

| Properties | Dynamic Lock | |
|---|---|---|
| MAC Address (Device Address) | ● | Insecure |
| Class of Device | ● | Insecure |
| RSSI | ● | Insecure |
| Link Establishment | ● | Insecure |
| Insecure Connection (SDP) (A Connection in Security Mode 4 - Level 0) | ● | Insecure |
| Secure Connection (e.g RFCOMM) (A connection in Security Mode 4 - Level 4) | ● | Secure |
| A Message over RFCOMM | X | Secure |

**Слика 2.7.2. Сигурност атрибута послатих кроз огласни пакет кроз сигурносне модуле 0 и 4[3]**

# 3. СИМУЛАЦИЈА REPLAY НАПАДА

Поглавље је посвећено једном сценарију злоупотребе *Bluetooth Low Energy* протокола, познатом као *replay*[12][13] напад. У овом сценарију злоупотребе пакети оглашавања BLE протокола (*beacons*) једног уређаја типа паметног телефона се ослушкују и чувају на другом уређају. Паметни телефон чији су пакети оглашавања сачувани се назива уређај-жртва, а особа која је власник овог паметног телефона је корисник-жртва. Уређај који ослушкује и чува *beacon* уређаја жртве је уређај-нападач, а особа која контролише акције овог уређаја је нападач-особа. Злоупотреба протокола настаје у тренутку када трећи уређај (на пример централа, поглавље 2.3.1) жели да комуницира са уређајем-жртвом, или жели да сазна информације о уређају-жртви кроз *beacon* податак: уређај жртва је угашен или ју је корисник-жртва физички однела далеко, али подаци се шаљу (поново пуштају, енглески *replay*) са уређаја-нападача и централа, на основу тих података, детектује да је уређај-жртва близу и присутна, иако заправо физички није. На овај начин уређај-нападач може пружити особи-нападачу приступ претходно откључаним просторима и уређајима (паметне браве) који се нису закључали пошто је уређај-нападач опонаша уређај-жртву.

Сигурносни проблем у првим верзијама *Dynamic lock* функционалности је сличан *replay* нападу, и овај пројекат имплементира симулацију овог напада као демонстрацију проблема *Dynamic lock*-а дискутовану раније. Имплементирани пројекат нема имплементацију сигурносног решења из поглавља 2.7.

Током поглавља 3, биће објашњени кораци симулације *replay* напада кроз имплементирани пројекат, затим ће бити анализирани токови и детаљи саме имплементације решења, и одлике коришћеног уређаја-нападача.

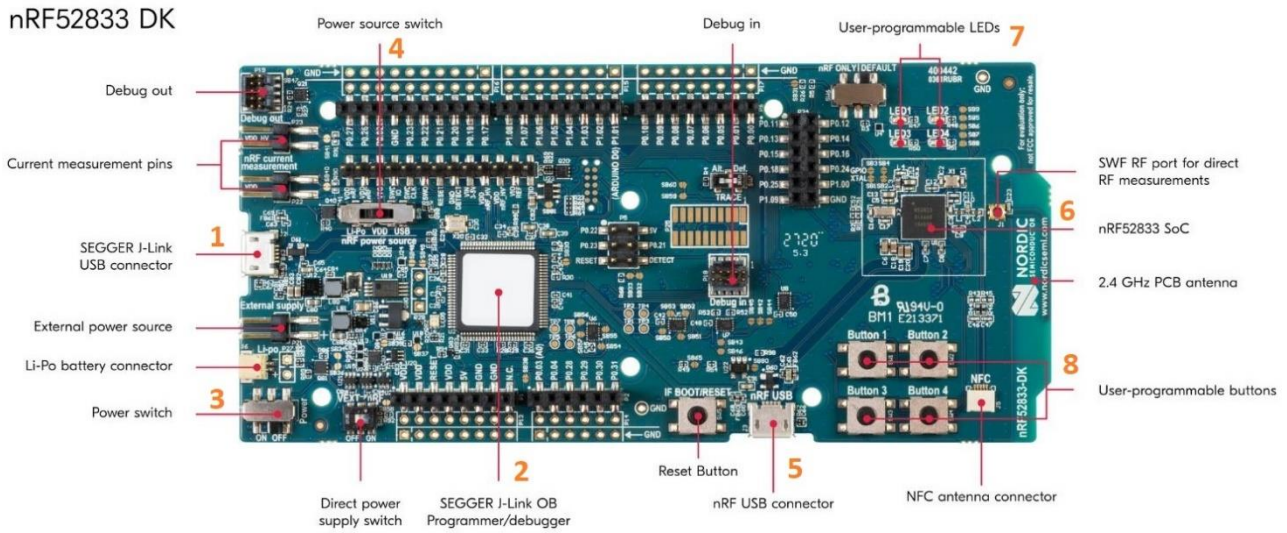
## 3.1. Уређај Nordic nRF52833 и Zephyr

Уређај nRF52833 је производ компаније *NordicSemiconductor*, из серије nRF52, решење за програмирање разноликих протокола безжичне комуникације: BT, BLE, *BT mash, Near Field Communication NFC, OpenThread, Zigbee*.[4] Коришћени уређај је хардверска прва верзија, ревизија 1.0.0 одобрена јуна 2020. *System on Chip* покреће *Zephyr OS*, који је препоручен од стране SIG организације за рад и истраживање на тему BLT.[11]. Ово је било кључно у избору уређаја за пројекат који се бави сигурношћу протокола.

*Zephyr RTOS* пружа већ имплементирану подршку и услужне библиотеке за једноставан рад, али и за дубоко модификовање и потпуну програмерску контролу над истим. Неки од подржаних стекова: BLE, BLE mash, WiFi, TCP/IP, *Trusted Firmware-M, MCUBoot – secure bootloader*).

### *3.1.1 Преглед делова уређаја*

На слици 3.1. се налазе ознаке свих делова уређаја.




**Слика 3.1. Преглед релевантних компонената плочице nRF52833.**

Компоненте од значаја:

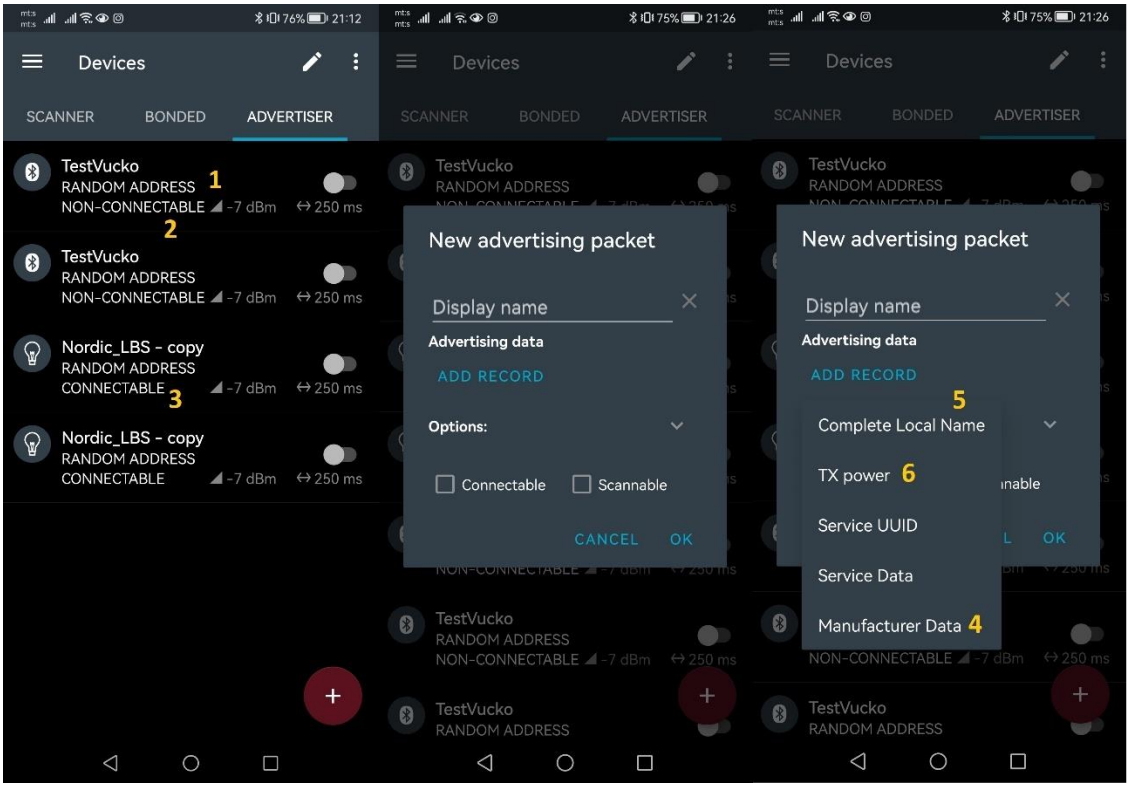
- *SEGGER J-Link USB connector* - (1) коришћен *Micro* USB конектор као извор струје за уређај. Извор енергије је (препоручено) USB 2.0. Алтернативно се може користити и (5). Разлог употребе баш овог извора је *Segger* профајлер и дебугер који ради и директно претвара струју у лаган програмерски излаз, са дијагностиком.
- *Power switch* – (3) пуштање струјног извора у остатак уређаја.
- nRF52833 SoC – (6) локација физичког чипа
- *User-programable LEDs* - (7) 4 лампице доступне за програмирање (упис вредности у)
- *User-programable buttons* – (8) 4 дугмета доступна за програмирање (читање вредности из)

## 3.2. Конфигурација жртве напада на паметном телефону

У сценарију напада на *Dynamic lock* жртва напада је особа чији је уређај (паметни телефон) нападнут, и *beacon* пакети снимљени и касније оглашавани од стране другог, злонамерно конфигурисаног уређаја. Током симулирања напада у овом пројекту, користиће се апликација за мобилни телефон кроз коју се могу модификовати многи параметри. Кроз коришћење ове апликације ће се видети које податке сам оперативни систем *Android* не дозвољава да се бирају и ручно уносе, и самим тим како се имплементира заштита на нивоу оперативног система од неких типова злоупотребе уређаја на коме ради тај OS. За имплементирану симулацију није неопходна ова апликација.

*Nordic* пружа мобилну апликацију за паметне телефоне доступну на званичним продавницама апликација, под називом "*nRF Connect for Mobile" by Nordic Semiconductor ASA*. Коришћена верзија је 4.28.1. Ова апликација ће бити коришћена да би се заобишао

системски генерисан beacon уређаја и пустио у оглашавање додатни, који ће се користити за тест, и који је имплементација дискусије из поглавља 3.3.1.




**Слика 3.2 Кораци креирања произвољног пакета за оглашавање (*beacon*).**

Анализа кључних корака и одлука тока конфигурације *beacon* пакета који ће се слати са паметног уређаја, а који је креиран кроз апликацију:

- Android оперативни систем увек генерише MAC адресу, зато је увек ознака да је адреса случајна (3.2. 1)
- *Beacon* може представљати чист оглашивач - уређај (3.2. 2) или потенцијално упарив (3.5. 3). Да се што ближе опонаша прави уређај типа паметног телефона, поставља се поље на опцију потенцијално упарив.
- За тест и лакшу идентификацију (слике 3.2. 4) подешава се неуобичајни произвођача.[5] У току скенирања простора у јако прометном окружењу (жкола, канцеларија) ће бити много *Apple* и *Google beacon*-а.

## 3.3. Кораци симулације злоупотребе

### *3.3.1. Конфигурација beacon-а жртве-уређаја*

Током симулирања напада у овом пројекту, користиће се апликација за мобилни телефон кроз коју се може направити пакет за оглашавање произвољним параметрима који се тичу имена уређаја (или празно име), подаци о произвошачу, и типови оглашавања. Ова

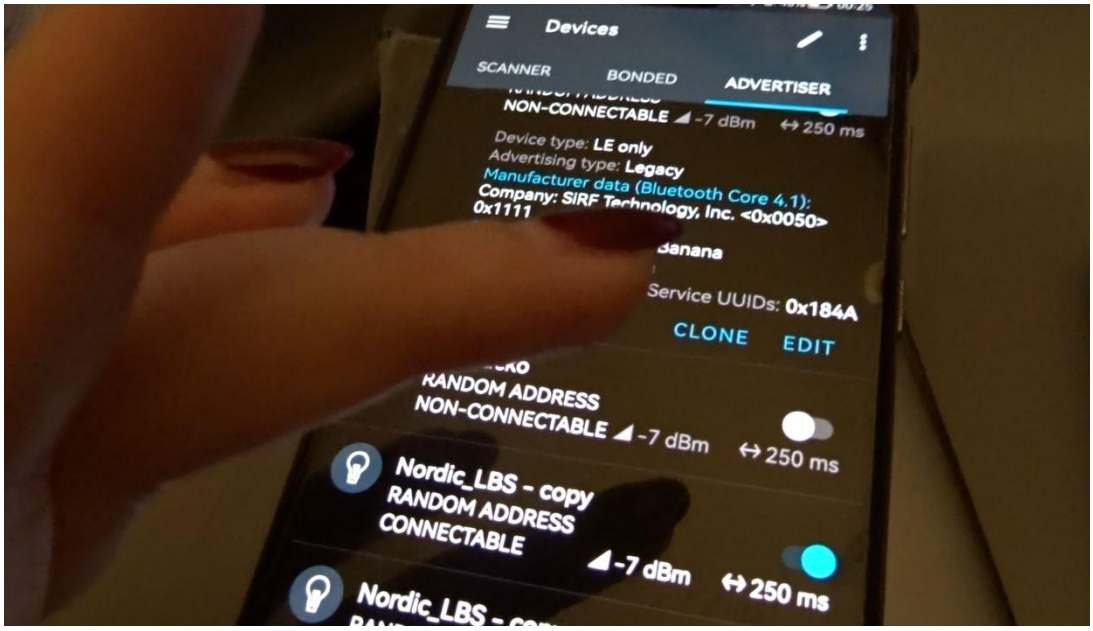
апликација ће се налазити на стварном физичком паметном телефону, који ће бити у улози уређаја-жртве током симулације.

*Android* OS не дозвољава да било који апликативни део диктира MAC адресу, већ је сам генерише. Због тога, у примеру креираног пакета за оглашавање са слике 3.3.1.1 под именом “Nordic_LBS - copy” постоји податак RANDOM ADDRESS. Ово је начин којим се овај оперативни систем брани од потенцијално злонамерних корисника, и апликација.

Да би се репродуковала ситуација у којој мобилни телефон пасивно оглашава своје присуство, кроз *beacon* без имена, тако се подешава и овај пакет. Постоји опција да је име уређаја празно.

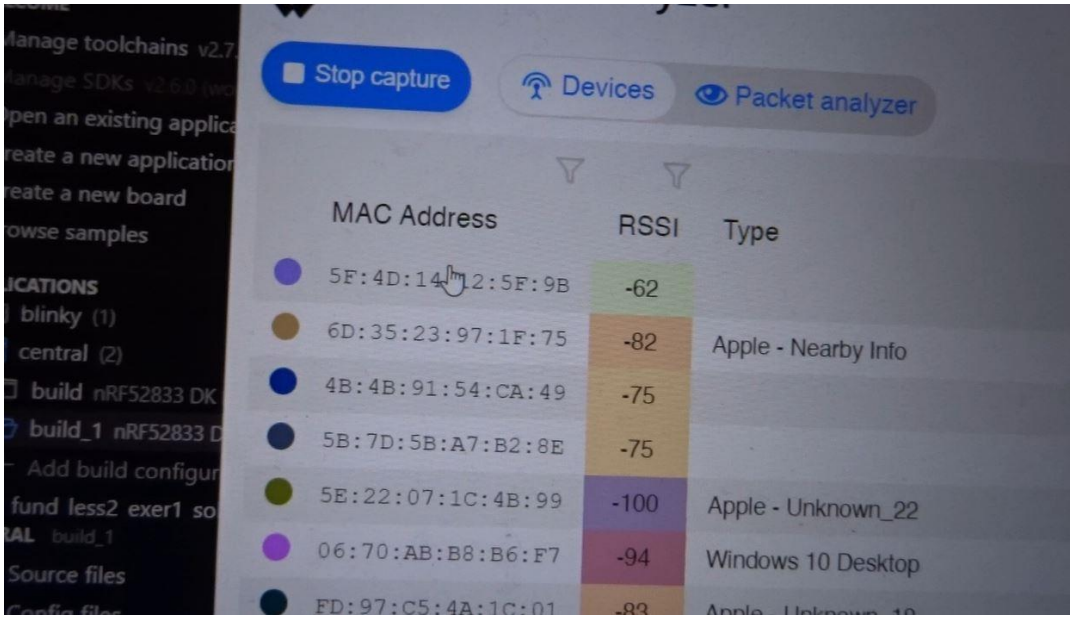
Пошто је у тренутку креирања пакета адреса унутар њега непозната, а име уређаја је празно, ради тестирања и лакшег праћења и идентификације пакета постављен је UUID произвођача на 0x0050[5]. Сви *beacon* пакети на овом паметном телефону су конфигурисани са истим произвођачем, који је био довољно јединствен да се истакне у скенираном тестном окружењу. Овај податак се види на слици 3.3.1.1, за један пакет у листи.

Од додатних важних параметара, важно је да је *beacon* потенцијално повезив, и да је типа Low Energy или неки LE хибрид. У коришћеном пакету конфигурисан је пакет за ниску потропњу.




**Слика 3.3.1.1 Почетак емитовања упаривог уређаја Nordic_LSB – copy**

Корисник у улози нападача помоћу алата за праћење детектује MAC адресу пакета који је циљ напада, на основу времена почетка оглашавања и конфигурисаног произвођача. У анализираном случају, адреса жртве је 5F:4D:14:12:5F:9B. Ово није практично за прави напад коме је Dynamic Lock био подложан у првој верзији. За овај пројекат део препознавања жртве уређаја је јако упрошћен, и стављен у други план, док је предност дата анализи и самом replay нападу.

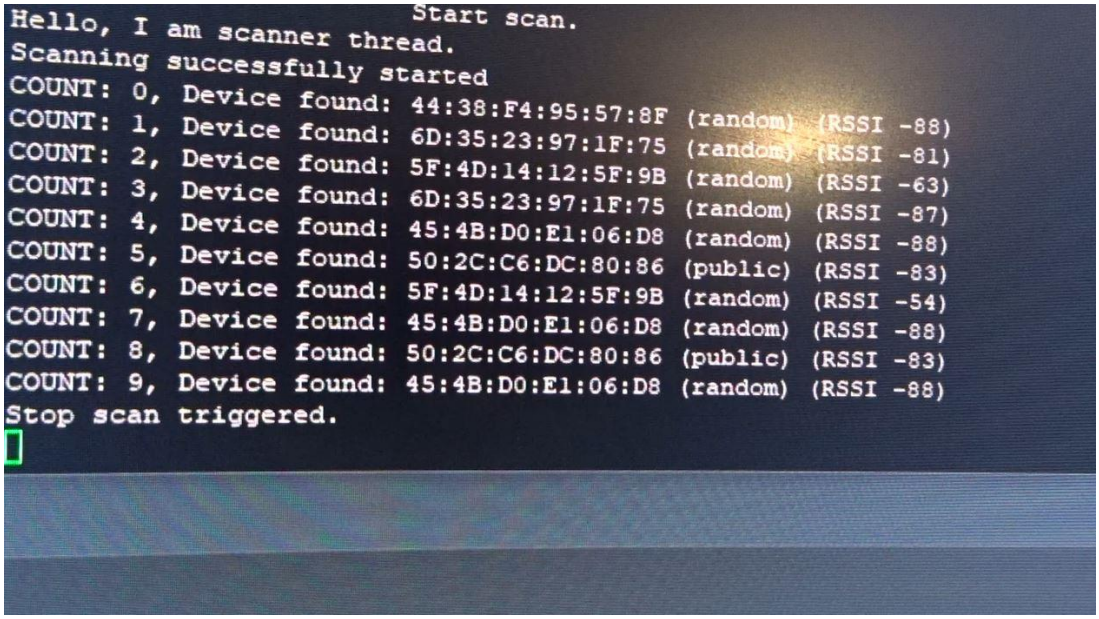



**Слика 3.3.1.2 Препознавање адресе жртве напада**

### *3.3.2. Фаза скенирања нападача-особе кроз нападач-уређај*

У овој фази нападач-особа уочава да је жртва-особа присутна у блиском простору са жртвом-уређајем, и нападач-особа почиње кораке скенирања (ослучкивања) у потрази за beacon подацима уређаја-жртве. Нападач-особа покреће скенирање притиском дугмета на уређају nRF52833 ( ово је уређај-нападач, више о уређају се налази у поглављу 3.1.), ради се скенирање 10 пакета (конфигурабилни број пакета), који није филтриран, и може се десити да се у интервалу скенирања, када траје дуже, буду ухваћена два или више *beacon*-а истог уређаја. На слици 3.1.2 се види пакет од интереса са адресом 5F:4D:14:12:5F:9B, ухваћен је два пута, под индексима 2 и 6, са различитим RSSI, што потврђује да су ово beacon са уређаја-жртве који је оглашен два пута у интервалу скенирања уређаја-нападача.

Када је 10 пакета нађено и снимљено на уређају-нападачу, скенирање се паузира. Ово је имплементирано кроз независну нит (независну од главне и *idle* нити уређаја) која се покреће при стартовању уређаја-нападача, и која се повремено буди да провери да ли је у систему стање читања.



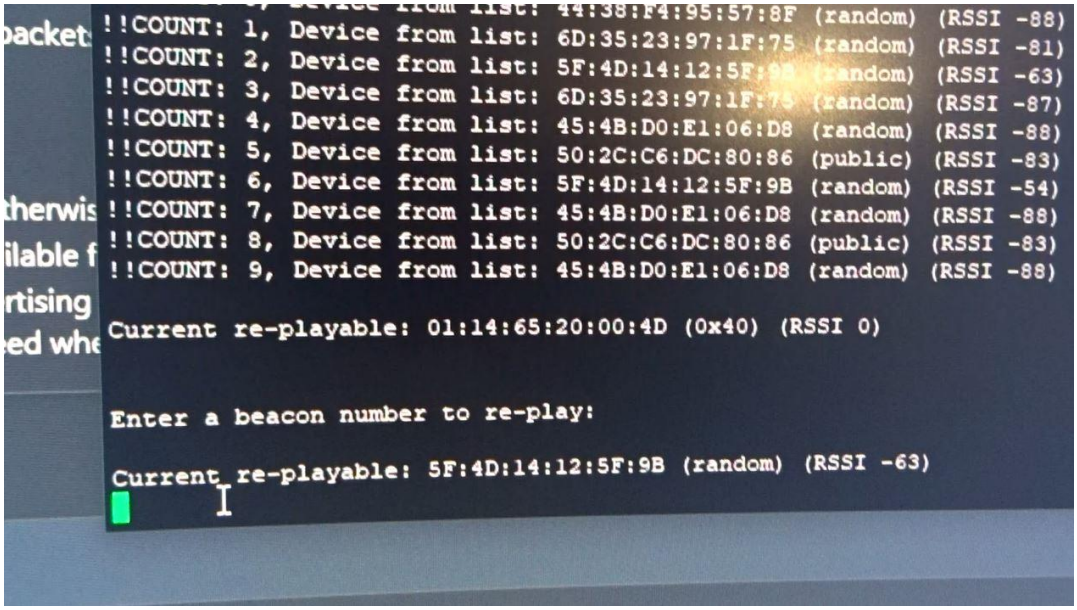

**Слика 331.2 Скенирање 10 пакета**

Стање читања поставља системски прекид (*interrupt*) имплементиран над прекидачем за притисак дугмета на уређају-нападачу. Једини посао ове прекидне рутине је постављање система на стање скенирања.

### *3.3.3. Фаза селекције нападача-особе*

Мора се извршити бар једно скенирање система. Када корисник активира дугме на уређају nRF52883, улазимо у другу прекидну рутину која позива *callback* функцију, а та функција врши излиставање последњих снимљених 10 пакета-*beacona,* и пружа особи-нападачу опцију да помоћу терминала унесе број између 0 и 9 да би тако изабрао пакет који жели да шаље као свој (*replay*). У примеру се жељени пакет налази на позицији 2 и 6. Када је пакет изабран, пакет се снима у посебно место у меморији уређаја-нападача.

При терминал комуникацији особе-нападача и уређаја-нападача користи се серијски *Universal Asynchronus Reciever-Transmitter* UART комуникациони порт, у једном смеру (терминал ка уређају-нападачу). Разлог једносмерности је коришћење терминала као излаза конфигурисаног на *System on Chip*-у овог уређаја, детаљније објашњеног у поглављу 3.4.



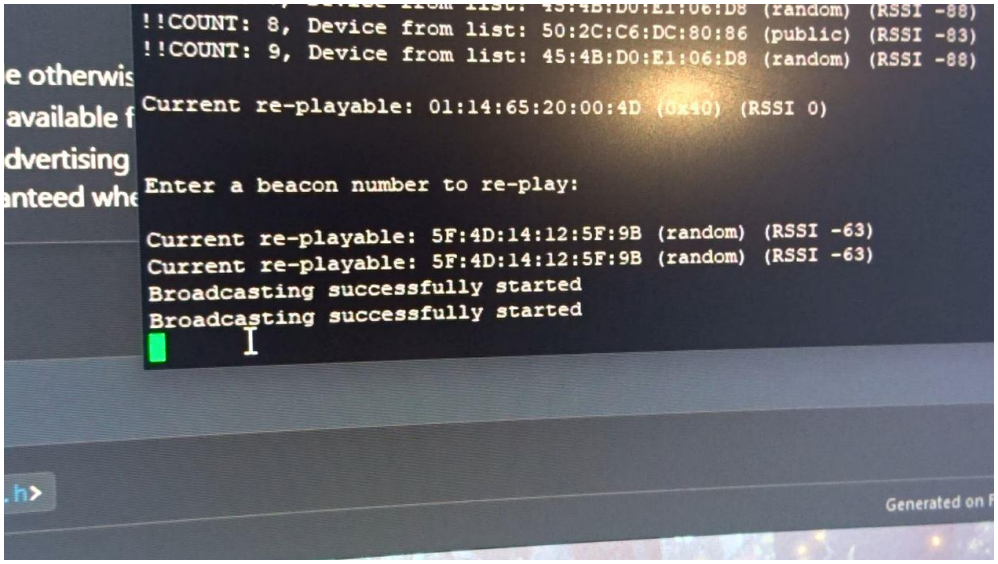

**Слика 3.3.3 Пакет је изабран**

Особа-нападач може више пута да листа снимљене пакете и бира пакет који ће се слати, и може више пута иницирати скенирање. У сценарију са слике 3.3.3. пакет од интереса 5F:4D:14:12:5F:9B се налази на позицијама са индексом 2 и 6. За овај пројекат може се изабрати било који од ова два пакета, пошто су RSSI вредности приближне, а граница која је постављена за ознаку да је уређај недоступан је -100dB – ово је одлика алата коришћеног за праћење пакета описаног у поглављу 3.5.

### *3.3.4. Replay beacon-а са уређаја-нападача*

У овом кораку *beacon* мора бити изабран. После притиска дугмета на уређају-нападачу, улази се у трећу прекидну рутину, чији је посао да стави систем у стање оглашавања.

Друга нит, конфигурисана на сличан начин као нит скенирања, почиње са оглашавањем изабраног *beacon*-а (претходно снимљеног у посебну промењиву на уређају-нападачу) и улази у бесконачно емитовање истог. Ово је крај рада особе-нападача са уређајем-нападачем, и да би се цео систем поново покренуо, потребно је ресетовати уређај-нападач.

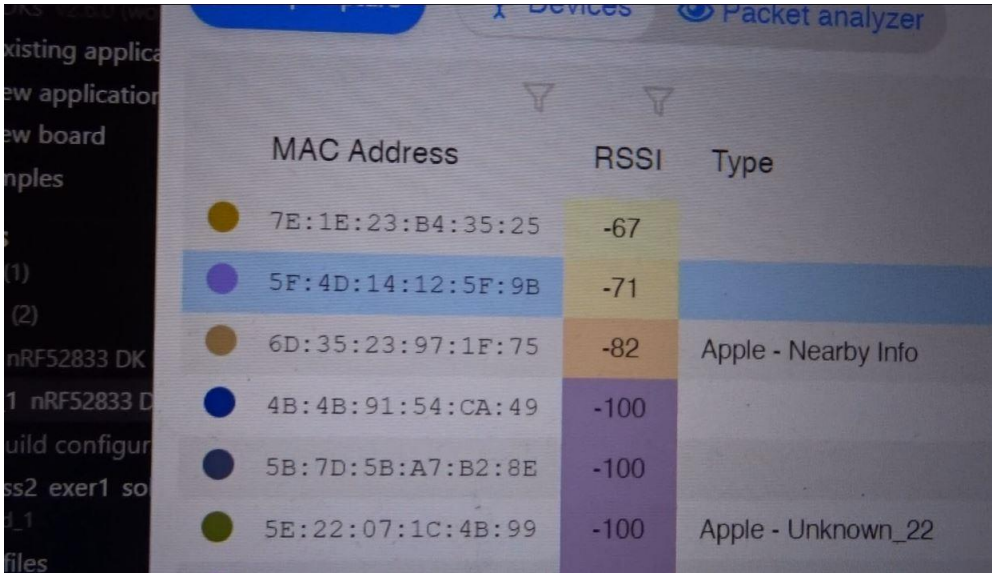



**Слика 3.3.4 Изабрани пакет се сада оглашава са уређаја-нападача**

### *3.3.5. Доказ успешне симулације напада*

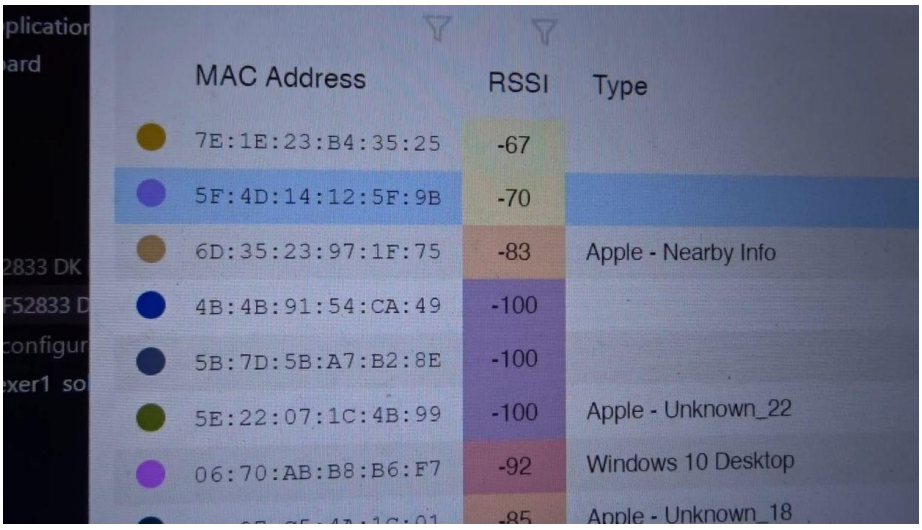
Наставак теста, након почетка емитовања овог *beacon*-а је да се жртва-уређај удаљи или угаси. Тако се симулира сценарио да је жртва-корисник устала, и физички удаљила од рачунара. Очекује се да се, у стварном свету, региструје да је жртва довољно далеко и да се рачунар закључава са *dynamic lock*-ом – Детекција удаљености је кључна. За потребе верификације успешности злоупотребе *dynamic lock-a* се не користи права функционалност са оперативног система, пошто су верзије оперативног система *Windows* 10 у коме се налазе прве верзије ове функционалности недоступне за преузимање и инсталирање. Ради праћења раздаљине између уређаја-жртве и централе ће се користити апликација која посматра и мери RSSI вредности из пакета оглашавања уређаја-жртве, и након тога уређаја-нападача.

Ради верификације, у овом тесту се покреће још један beacon на мобилном уређају (слика 3.3.5.1), 7E:1E:23:B4:35:25. Подешавањем и посматрањем још једног *beacon*-а са уређаја-жртве постижемо верификацију да је *Bluetooth* заиста (физички) угашен. Очекујемо да ће пакет са адресом 7E:1E:23:B4:35:25 престати да се емитује када се *Bluetooth* буде угасио на уређају-жртви. Такође, очекујемо да ће пакет са адресом 5F:4D:14:12:5F:9B бити видљив и доступан и када *Bluetooth* буде угашен на урешају-жртви, пошто се сада емитује и са уређаја-нападача.



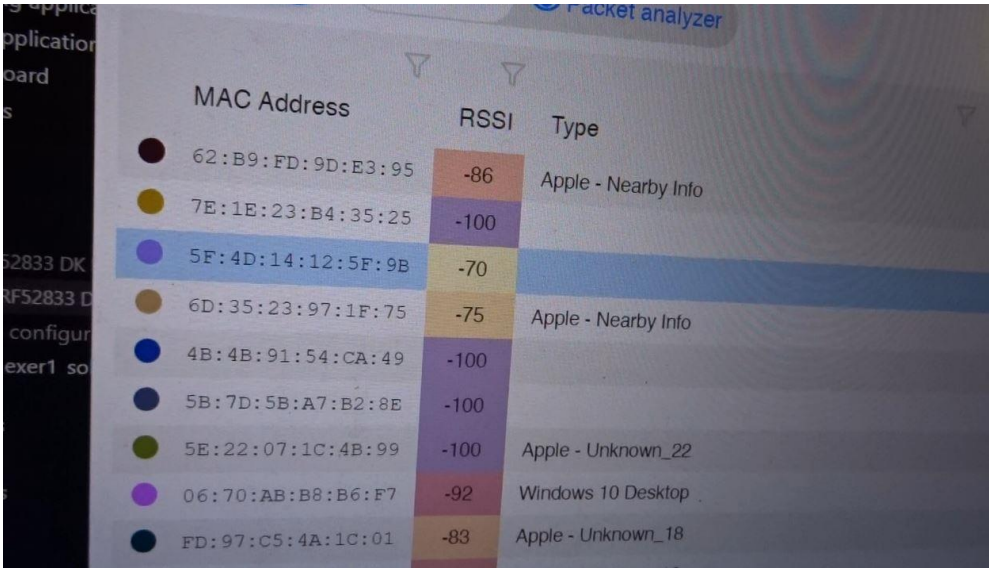

**Слика 3.3.5.1 Укључивање другог сигнала са истог физичког уређаја**

Након потврде да се нови, помоћни beacon види на систему за праћење, симулира се удаљавање уређаја-жртве и првог компромитованог пакета 5F:4D:14:12:5F:9B гашењем емитовања тог пакета. (слика 3.3.5.2) Начин гашења је супротан од начина паљења са слике 3.1.1.1. У овом тренутку се очекује да се пакет 5F:4D:14:12:5F:9B види као недоступан јер се више не емитује, али пошто је пакет компромитован и сада се емитује са уређаја-нападача, и даље се види као доступан.




**Слика 3.3.5.2 Изабрани пакет се више не емитује са паметног телефона**

Последњи корак верификације је да се физички угаси Bluetooth опција на паметном телефону у потпуности. Резултат ове операције (слика 3.3.5.3) показује да је помоћни *beacon* 7E:1E:23:B4:35:25 недоступан, што је очекивано. Очекује се да је и 5F:4D:14:12:5F:9B приказан као недоступан, али пошто се емитује са уређаја-нападача, *beacon* је у домету.




**Слика 3.3.5.3 Мобилни уређај је физички угасио bluetooth опцију**

У симулацији се види да је жртва 5F:4D:14:12:5F:9B видљива на слици 3.3.5.3 - снага емитовања овог beacon-а није постала -100dB. Снага емитовања пакета са адресом 7E:1E:23:B4:35:25 је постала -100dB, што је очекивано.

Из угла посматрања алата за скенирање *beacon* 5F:4D:14:12:5F:9B је доступан и оглашава се, и то за овај алат значи да је и уређај који емитује овај пакет, уређај-жртва, упаљен (*Bluetooth* на уређају ради и емитује пакете). То није тачно пошто је уређај-жртва угашен (*Bluetooth* је угашен). Уређај-нападач се успешно претвара да је он уређај-жртва, из перспективе алата који посматра пакете.

## 3.4. Анализа кода

### *3.4.1. Конфигурациони параметри пројекта*

Програмирање се врши на екстерном уређају па су ресурси физички ограничени или тешко надоградиви. Важно је да се у конфигурационим документима назначи свака функционалност и модел рада који се користи, као и да се дефинише сваки елемент уређаја који се користи (дугмад, лампица, и слично) (видети прилог А.1). За потребе овог пројекта потребно је укључити *Bluetooth* компоненту на уређају, *Bluetooth Broadcaster* модел рада, и *Bluetooth Observer* мод рада.

За комуникацију са периферијама уређаја (дугмад и лед) користи се *General Purpose Input Output* (GPIO) , и библиотека *Zephyr* DK, која је слој који даје једноставне функције за контролу истих.

Такође, да би се извршила комуникација корисника са уређајем преко терминала, користити се серијски UART порт, који се и декларише за употребу. Пошто акција клијента није блокирајућа, нити од кључног значаја за систем, биће асинхрона комуникација.

### *3.4.2. Главна функција*

Zephyr RTOS не захтева дефиницију *main* функције. При покретању уређаја оперативни систем креира две нити - једна пролази и хардверски иницијализује цео систем, креира додатне нити за системске прекиде који су у беспосленом стању. Када прва нит заврши са иницијализацијом, стартује оперативни систем и стартује све друге дефинисане нити од стране програма, затим тражи главну функцију. Ако главна функција није дефинисана, главна нит иде у беспослено стање и не буди се до краја рада уређаја.

У главној функцији пројекта се мора извршити иницијализација елемената уређаја, у наставку - периферија. Прво, уређај се поставља у стање иницијализације помоћу промењљиве *system_state*, за случај да се неки системски прекид детектује. За исправан рад уређаја мора се обезбедити неометана иницијализација. Следећи корак је иницијализација контејнера у коме ће се сместити скенирани *beacon* пакети.

Након покретања сваке појединачне периферије мора се проверити да ли је дошло до неке грешке, да ли је било неких хардверских потешкоћа због којих главна нит у претходној фази није могла да га покрене, или све потребне напомене у конфигурацији нису дате. Ако је уређај спреман, ради се конфигурација са жељеним параметрима и моделом рада (дугме је периферија из које подаци улазе у уређај, тачније уређај прима информацију од акције над дугметом, а лампица је периферија на коју излази програмирани податак, тј уређај шаље команде лампици). Ово се ради за свако коришћено дугме и лампицу, и користи се *gpio_pin_configure*_dt из CONFIG_DK_LIBRARY.

UART је мало другачији, код њега такође проверава да ли је периферија спремна, затим се конфигуришу позиви за податке које добијају кроз тај серијског порта кроз *callback* и *callback_cb*. серијски порт се подешава као уређај за читање са (*uart_rx_enable – read enable*).

Следећи корак је покретање *Bluetooth* са позивом *bt_enable* без параметара – ово се користи када ће уређај у неком тренутку радити скенирање за пакетима. Ако је уређај искључиво за трансмисију пакета, као параметар се прослеђује функција која иницијализује beacon за трансмисију, и док се не позове *bt_disable*, уређај искључиво ради варијације трансмисија или конекцију, али никад скенирање.

На крају се иницијализују прекидне рутине, које се објашњавају детаљније у поглављу 3.4.4.

### *3.4.3. Иницијализација нити*

Нити се дефинишу испод главне функције:

```
K_THREAD_DEFINE(thread0_scanner, STACKSIZE, thread0, NULL, NULL, NULL,
THREAD0_PRIORITY, 0, 0).
```

Први параметар је идентификатор нити, који може да се прати током анализирања проблема. Следи параметар о е кориснички дефинисаној величини стека и приоритет ове нити.

```
// Thread statuses and variables
#define STACKSIZE 1024
#define THREAD0_PRIORITY 7
```

О приоритету нити – Приоритети нити могу бити позитивне и негативне. Најприоритетнија нит за извршавање је она са најнижим бројем – главна нит је са приоритетом 0, а *idle* са највишим бројем (15, ако другачије није наведено у конфигурацији за број корисничких нити). Нити могу имати и негативан број (оне су везане за детекцију системских прекида) и када негативна нит ради у језгру процесора, не напушта процесорско време док не заврши сав посао. Нити са позитивним и нултим приоритетом су доступне за *rescheduling*.

### *3.4.4. Иницијализација прекидних рутина*

Следи иницијализација 3 коришћена дугмета за системске прекиде. Свако дугме добија свој прекид и своју функцију за обраду рутине у виду, повезану кроз *callback* функцију и *callback_cb* подацима, које у овом пројекту неће користити.

```
/* Button0 interrupt configurations */
err = gpio_pin_interrupt_configure_dt(&button0,
GPIO_INT_EDGE_TO_ACTIVE );
gpio_init_callback(&button_cb_data, button_pressed, BIT(button0.pin));
gpio_add_callback(button0.port, &button_cb_data);
```

За тачне адресе периферија користе се алиаси (поглавље 3.5.). Позивом макроа DT_ALIAS извлачи се адреса за периферију без икакве потребе да програмер види хекс облик адресе.

```
#define SW0_NODE   DT_ALIAS(sw0)
static const struct gpio_dt_spec button0 = GPIO_DT_SPEC_GET(SW0_NODE,
gpios);
```

У приложеном примеру на притисак дугмета 0 се позва *button_pressed* функцију која само поставља статус система на стање скенирања. Јако слична операција се извршава и када се притисне дугме 3, у *button_broadcast* функцији. Током интеракције са дугметом 1 извршава се испис за корисника о упутству за изабир пакета. UART периферија кроз дефинисане функције ради ослушкивање и прикупљање корисничког уноса који може да се деси и касније.

Напомена – у функцији за селекцију beacon-a, *button_pressed_chose*, постоји позив функцији *bt_addr_le_to_str* - ово је само функција за парсирање пакета у стринг облик.

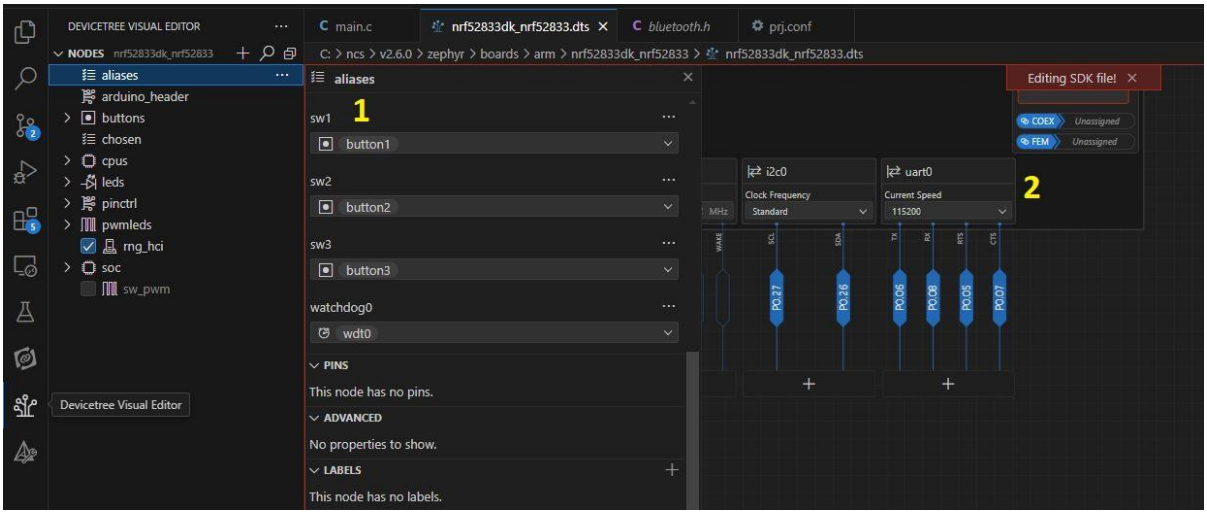
### *3.4.5. Серијска комуникација*

За разлику од дугмади, UART заправо користи пакет прослеђен кроз *callback* кроз *uart_cb*. У функцији се извршава примитивна серијализација и детекција одабира корисника. На основу парсиране вредности, извршава се плитко копирање индексираног пакета из листе 10 скенираних пакета, у променљиву *selected_beacon*. Подаци из ове променљиве се касније користе за трансмитовање.

## 3.5. Visual Studio Code и nRFConnect екстензија за писање кода

За писање програма и контролу уређаја користи се Visual Studio Code (верзија 1.93.1) програмско окружење, и екстензија nRFConnect (верзија SDK 2.6.0, toolchain 2.7.0), доступна у продавници екстензија VSCa.

Из самог радног окружења, од значаја је *Devicetree Visual Editor*-а. У картици *Nodes* се налази *aliases* (слика 3.5 - 1) – ово је листа коју *Zephyr* интерно одржава, а у којој се може видети листа пре-дефиних кодове за приступ адресама компонената уређаја. У овом пројекту, примеру, за програмирање дугмета 1 користиће се алиас *sw1* (слика 3.5 - 2).




**Слика 3.5 Ток конфигурације кода за циљану платформу.**

## 3.6. Алати за мониторисање Bluetooth пакета

За потребе овог пројекта, и планирани низ корака које ће се извести, праћење јачине сигнала уређаја кроз RSSI вредност ће се вршити путем програма Acrylic Bluetooth LE Analyzer. Коришћење овог програма није обавезно, и као алтернатива за процес праћења beacon-а се може користити мобилни телефон са апликацијом из поглавља 3.5, која такође приказује податак о RSSI сигналу, и јако га ажурно освежава.

*Acrylic Bluetooth LE Analyzer*[14] је лиценцирани алат за мониторисање BLE. Даје опције за хватање свих пакета, јако лако и без оптерећења система хвата све *beacon*-е и, док ради у позадини или на другом монитору, има опцију за звучно обавештење о новом ухваћеном уређају и *beacon*-u.

## 3.7. Дискусија решења

Систем није оптимизован за поновну употребу, потребно је физички рестартовати уређај-нападач. При имплементацији решења је посматран напад у стварном животу, који се састоји од особе-нападача који бира особу-жртву, седа у близини те жртве, скенира, детектује и ретрансмитује пакет, и када особа-жртва одлази заједно са уређајем-жртвом,

особа-нападач физички седа за рачунар жртве који је овом акцијом остао откључан. У том сценарију, мета нападача је један рачунар. Из угла оптималног писања програмера, генерално, сваки пројекат би требало да има опцију да се понови без физичког ресета машине.

Следећи детаљ би била организација пројекта. Током развоја организација је разматрана, али су због статичности све функције биле декларисане и дефинисане у заглављу, без .c фајлова. Такође, пошто је пројекат процедуралан, делови се не могу потпуно изоловати и постоји потреба за увлачењем заглавља у друга заглавља (пошто нема .c фајлова), и постаје јако нечитко прати ток рада. У изабраној организацији, лако је наћи и испратити све променљиве, функције, и дефиниције.

# 4. Закључак

Симулацијом сценарија злоупотребе огласних пакета, познатом као *replay* напад је показано како особа-нападач долази до огласних пакета уређаја жртве, и како су ти пакети успешно слати у простор. Када је уређај-жртва постао недоступан кроз гашење *Bluetooth*-а, и даље је уређај-жртва био видљив у једноставном скенеру на трећем уређају (уређај улози централе-скенера). Пошто је трећи уређај радио једноставно мапирање уређаја у своју базу са кључевима MAC адреса и име уређаја, доказано је да овај начин идентификације није довољно сигуран начин да се уређај-жртва идентификује као физички присутна у простору. Ова комуникација је рањива на replay тип напада. У случају функционалности *Dynamic Lock*, рањивост на *replay* напад је била уско повезана са стањем закључавања рачунара особе-жртве, и кроз симулирани сценарио је приказана раљивост првих верзија *Dynamic Lock*-а.

*Bluetooth* стандард је присутан  јако дуго, и безбедносне мере које се у њему користе су разноликог нивоа, и јако поуздане. BLE, иако дели део техничког стека са BT, има нове проблематике и ситуације за коришћење. У детаљно анализираном примеру напада решење је упит ка извору трансмитовања за пакете који садрже енкриптоване податке, пошто је то једини поуздан начин да будемо сигурни у интегритет пошиљаоца (емитера пакета) и интегритет податка.

При коришћењу BLE протокола, за разлику од BT протокола, треба обратити додатну пажњу на ситуације у којима емитери пакета одбијају захтеве за додатне информације или повезивање (одашиљачи), па се у конструисање система мора више и дубље размислити о томе од кога примамо информације и клико нам је важан интегритет истих - доста безбедносних мера постоји, али је и заједница SIG отворена и активна у разматрању нових проблематика и решења.

# Литература


[1] *Bluetooth Pairing Part 1 – Pairing feature Exchange[Online]* https://www.bluetooth.com/blog/bluetooth-pairing-part-1-pairing-feature-exchange/ (17.09.2024.)

[2] *Bluetooth LE Advertising training [Online]* https://academy.nordicsemi.com/courses/bluetooth-low-energy-fundamentals/lessons/lesson-2-bluetooth-le-advertising/ (17.09.2024.)

[3] *BlueMaster: bypassing and Fixing Bluetooth-Based Proximity Authentication[Online]*:24-28, 41-55. https://pdfs.semanticscholar.org/b1a4/54e5daca91ae5fb97869e3fa163feac00946.pdf (17.09.2024.)

[4] *nRF52833 DK Product Brief Version 1.2[Online]*:1-2 https://www.nordicsemi.com/-/media/Software-and-other-downloads/Product-Briefs/nRF52833-DK-PB.pdf (16.09.2024.)

[5] *Bluetooth Assigned Numbers[Online]* https://www.bluetooth.com/specifications/assigned-numbers/ (16.09.2024.)

[6] *Guide to Bluetooth Security[Online]*:17-37 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-121r2-upd1.pdf (19.09.2024.)

[7] *Announcing Windows 10 Insider Preview Build 15031 for PC [Online]*, https://blogs.windows.com/windows-insider/2017/02/08/announcing-windows-10-insider-preview-build-15031-pc/ (19.09.2024.)

[8] *Dynamic Lock [Online]* https://learn.microsoft.com/en-us/windows/security/identity-protection/hello-for-business/hello-feature-dynamic-lock (19.09.2024.)

[9] *Bluetooth technology overview - Bluetooth Wireless technology [Online]* https://www.bluetooth.com/learn-about-bluetooth/tech-overview/ (19.09.2024.)

[10] *Bluetooth Specification and Documentation - Core 4.0 Specification [Online]* ,2165-2170 https://www.bluetooth.com/specifications/specs/core-specification-4-0/ (19.09.2024.)

[11] *An Introduction to Bluetooth LE Security Study Guide [Online]* https://www.bluetooth.com/blog/an-introduction-to-the-bluetooth-le-security-study-guide/ (19.09.2024.)

[12] *On Preventing Replay Attacks on Security Protocols [Online-Archived]*, 1-3 https://web.archive.org/web/20220120144617/https://apps.dtic.mil/dtic/tr/fulltext/u2/a462295.pdf (19.09.2024.)

[13] *Mathematical Problems in Engineering: Takagi-Sugeno Fuzzy Control for a Nonlinear Networked System Exposed to a replay Attack [Online]* https://onlinelibrary.wiley.com/doi/10.1155/2021/6618105 (19.09.2024.)

[14] *Bluetooth LE Analyzer – Analyse your Bluetooth Low energy devices [Online]*, https://www.acrylicwifi.com/en/bluetooth-analyzer/ (24.09.2024.)

# СПИСАК СКРАЋЕНИЦА

| | |
|---|---|
| API | *Application Programming Interface* |
| BLE | *Bluetooth Low Energy* |
| NFC | *Near Field Communication* |
| SoC | *System on Chip* |
| RTOS | *Real-Time Operating System* |
| COM | *Communication Port* |
| GPIO | *General Purpose Input Output* |
| BT | *Bluetooth* |
| RSSI | *Received Signal Strength Indicator* |
| SIG | *Special Interest Group* |

# СПИСАК СЛИКА

# СПИСАК ТАБЕЛА

# A. Изворни код пројекта

## A.1. proj.conf

```
CONFIG_BT=y
CONFIG_BT_BROADCASTER=y
CONFIG_BT_OBSERVER=y
CONFIG_GPIO=y
CONFIG_DK_LIBRARY=y
CONFIG_SERIAL=y
CONFIG_UART_ASYNC_API=y
```

## A.2. main.c

```
/* Scanner, beacon selector and re-player 09-2024 vj120086d@student.etf.bg.ac.rs
*/

// Zephyr/core libraries
#include <zephyr/types.h>
#include <stddef.h>
#include <errno.h>
#include <zephyr/kernel.h>
#include <zephyr/sys/printk.h>
// Bluetooth libraries
#include <zephyr/bluetooth/bluetooth.h>
#include <zephyr/bluetooth/hci.h>
#include <zephyr/bluetooth/conn.h>
#include <zephyr/bluetooth/uuid.h>
#include <zephyr/bluetooth/gatt.h>
#include <zephyr/sys/byteorder.h>
// Buttons, LED, and UART/console communitaction
#include <zephyr/device.h>
#include <zephyr/devicetree.h>
#include <zephyr/drivers/gpio.h>
#include <zephyr/drivers/uart.h>


// System states (user actions in progress)
#define STATE_INIT       0
#define STATE_SCAN       1
#define STATE_THINKING   3
#define STATE_BROADCAST  4

// Scan limmit (prototype)
#define MAX_SCAN 10

// Input data for console talk
#define RECEIVE_BUFF_SIZE 10
#define RECEIVE_TIMEOUT 100

// Scanned device structure
struct scanned_device{
```

```
	bt_addr_le_t *addr;
	int8_t rssi;
	uint8_t type;
	struct net_buf_simple *ad;
} scanned_dev[MAX_SCAN];

// Thread statuses and variables
#define STACKSIZE 1024
#define THREAD0_PRIORITY 7
#define THREAD1_PRIORITY 7


// System state variales
static unsigned int count_scans = 0;
static unsigned int system_state = STATE_INIT;
static struct scanned_device selected_beacon;

/* Peripherial initializations */

/* Button initializations */
#define SW0_NODE  DT_ALIAS(sw0)
static const struct gpio_dt_spec button0 = GPIO_DT_SPEC_GET(SW0_NODE, gpios);
#define SW1_NODE  DT_ALIAS(sw1)
static const struct gpio_dt_spec button1 = GPIO_DT_SPEC_GET(SW1_NODE, gpios);
#define SW2_NODE  DT_ALIAS(sw2)
static const struct gpio_dt_spec button2 = GPIO_DT_SPEC_GET(SW2_NODE, gpios);

/* LED initialization */
#define LED0_NODE DT_ALIAS(led0)
static const struct gpio_dt_spec led = GPIO_DT_SPEC_GET(LED0_NODE, gpios);

/* UART initialization */
const struct device *uart= DEVICE_DT_GET(DT_NODELABEL(uart0));

/* Peripherial initializations end */

/* BLE scanner */
static void start_scan(void);

static void device_found(const bt_addr_le_t *addr, int8_t rssi, uint8_t type,
                         struct net_buf_simple *ad)
{
	char addr_str[BT_ADDR_LE_STR_LEN];
	int err;

	/* We're only interested in connectable events -
	   any real mobile/smart phone is connectible.
	*/
	if (type != BT_GAP_ADV_TYPE_ADV_IND &&
	    type != BT_GAP_ADV_TYPE_ADV_DIRECT_IND) {
		return;
	}

	bt_addr_le_to_str(addr, addr_str, sizeof(addr_str));
	printk("COUNT: %d, Device found: %s (RSSI %d)\n", count_scans, addr_str,
rssi);

    // Save this data to the list
	memcpy(scanned_dev[count_scans].addr, addr, sizeof(bt_addr_le_t));
```

```
        scanned_dev[count_scans].rssi = rssi;
        scanned_dev[count_scans].type = type;
        memcpy(scanned_dev[count_scans].ad, ad, sizeof(struct net_buf_simple));

        count_scans++;

      if (count_scans >= MAX_SCAN)
        {
              bt_le_scan_stop();
              printk("Stop scan triggered.\n");
              return;
        }
}

static void start_scan(void)
{
        int err;
        count_scans = 0;

        err = bt_le_scan_start(BT_LE_SCAN_PASSIVE, device_found);
        if (err) {
              printk("Scanning failed to start (err %d)\n", err);
              return;
        }

        printk("Scanning successfully started\n");
}
/* BLE scanner end*/


static void start_broadcast(void)
{
        int err;

        err = bt_le_adv_start(BT_LE_ADV_NCONN, selected_beacon.ad, sizeof(struct
net_buf_simple), NULL, 0);
        if (err == ECONNREFUSED || err == ENOMEM ) {
              printk("Advertasing failed to start (err %d)\n", err);
              return;
        }

        printk("Broadcasting successfully started\n");
}

/* Thread - scanner */
void thread0(void)
{
        while (1) {
              k_busy_wait(1000000);
              if (system_state == STATE_SCAN){
            printk("Hello, I am scanner thread.\n");
                start_scan();
                system_state = STATE_THINKING;
              }
              k_yield();
        }
}

/* Thread - broadcaster */
```

```
void thread1(void)
{
	while (1) {
		k_busy_wait(5000);
		if (system_state == STATE_BROADCAST){
			start_broadcast();
		}
	}
}

/* Threads end*/

/* Button acction(s) */
void button_pressed(const struct device *dev, struct gpio_callback *cb, uint32_t
pins)
{
	printk("Hello, I am button interrupt.\n");

	// Test, give me a sign on board
    gpio_pin_toggle_dt(&led);

	if (system_state == STATE_THINKING || system_state == STATE_INIT){
		system_state = STATE_SCAN;
		printk("                Start scan.\n");
	}
	else {
	    printk("Not thinking time, no scan.\n");
	}
}

void button_pressed_choose(const struct device *dev, struct gpio_callback *cb,
uint32_t pins)
{
	printk("Hello, List of available beacons to re-play:.\n");

	// Test, give me a sign on board
    gpio_pin_toggle_dt(&led);

	// re-parse address to readable form
	char addr_str[BT_ADDR_LE_STR_LEN];

	for (int i = 0; i < MAX_SCAN; i++){
	    bt_addr_le_to_str(scanned_dev[i].addr, addr_str, sizeof(addr_str));
		printk("!!COUNT: %d, Device from list: %s (RSSI %d)\n",i, addr_str,
scanned_dev[i].rssi);
	}

	bt_addr_le_to_str(selected_beacon.addr, addr_str, sizeof(addr_str));
	printk("\nCurrent re-playable: %s (RSSI %d)\n\n", addr_str,
selected_beacon.rssi);


	printk("\nEnter a beacon number to re-play:\n\n");
}
void button_broadcast(const struct device *dev, struct gpio_callback *cb,
uint32_t pins)
{
	// Test, give me a sign on board
    gpio_pin_toggle_dt(&led);
```

```
	system_state = STATE_BROADCAST;

}

// UART callback

static uint8_t rx_buf[RECEIVE_BUFF_SIZE] = {0};
static void uart_cb(const struct device *dev, struct uart_event *evt, void
*user_data)
{
	switch (evt->type) {

	case UART_RX_RDY:

	if((evt->data.rx.len) == 1){

		if(evt->data.rx.buf[evt->data.rx.offset] == '0')
			memcpy(&selected_beacon, &scanned_dev[0], sizeof(struct
scanned_device));
		if(evt->data.rx.buf[evt->data.rx.offset] == '1')
			memcpy(&selected_beacon, &scanned_dev[1], sizeof(struct
scanned_device));
		else if (evt->data.rx.buf[evt->data.rx.offset] == '2')
			memcpy(&selected_beacon, &scanned_dev[2], sizeof(struct
scanned_device));
		else if (evt->data.rx.buf[evt->data.rx.offset] == '3')
			memcpy(&selected_beacon, &scanned_dev[3], sizeof(struct
scanned_device));
		else if (evt->data.rx.buf[evt->data.rx.offset] == '4')
			memcpy(&selected_beacon, &scanned_dev[4], sizeof(struct
scanned_device));
		else if (evt->data.rx.buf[evt->data.rx.offset] == '5')
			memcpy(&selected_beacon, &scanned_dev[5], sizeof(struct
scanned_device));
		else if (evt->data.rx.buf[evt->data.rx.offset] == '6')
			memcpy(&selected_beacon, &scanned_dev[6], sizeof(struct
scanned_device));
		else if (evt->data.rx.buf[evt->data.rx.offset] == '7')
			memcpy(&selected_beacon, &scanned_dev[7], sizeof(struct
scanned_device));
		else if (evt->data.rx.buf[evt->data.rx.offset] == '8')
			memcpy(&selected_beacon, &scanned_dev[8], sizeof(struct
scanned_device));
		else if (evt->data.rx.buf[evt->data.rx.offset] == '9')
			memcpy(&selected_beacon, &scanned_dev[9], sizeof(struct
scanned_device));

		}

	break;
	case UART_RX_DISABLED:
		uart_rx_enable(dev ,rx_buf,sizeof rx_buf,RECEIVE_TIMEOUT);
		break;

	default:
		break;
	}
```

```
	char addr_str[BT_ADDR_LE_STR_LEN];
	bt_addr_le_to_str(selected_beacon.addr, addr_str, sizeof(addr_str));
	printk("Current re-playable: %s (RSSI %d)\n", addr_str,
selected_beacon.rssi);
}


static struct gpio_callback button_cb_data;
static struct gpio_callback button_cb_data1;
static struct gpio_callback button_cb_data2;

/* Button end */


int main(void)
{
	system_state = STATE_INIT;
	int err;

	for (int i = 0; i < MAX_SCAN; i++){
		scanned_dev[i].addr = malloc(sizeof(bt_addr_le_t));
//		scanned_dev[i].addr = 0;
		scanned_dev[i].rssi = 0;
		scanned_dev[i].type = 0;
		scanned_dev[i].ad = malloc(sizeof(struct net_buf_simple));
	}

	/* Peripherial initializations */
    /* Configure Button 0 - scanner control */
	if (!device_is_ready(button0.port)) { return -1; }
	if (gpio_pin_configure_dt(&button0, GPIO_INPUT) < 0) { return -1; }

	/* Configure Button 1 - user beacon re-play selection control */
	if (!device_is_ready(button1.port)) { return -1; }
	if (gpio_pin_configure_dt(&button1, GPIO_INPUT) < 0) { return -1; }

	/* Configure Button 2 - re-play beacon */
	if (!device_is_ready(button2.port)) { return -1; }
	if (gpio_pin_configure_dt(&button2, GPIO_INPUT) < 0) { return -1; }

    /* Configure LED - test control */
	if (!device_is_ready(led.port)) { return -1; }
	if (gpio_pin_configure_dt(&led, GPIO_OUTPUT_ACTIVE) < 0) { return -1; }

	/* UART setup */
	if (!device_is_ready(uart)) { return -1 ;}
	if (uart_callback_set(uart, uart_cb, NULL)) { return -1; }
	if (uart_rx_enable(uart ,rx_buf,sizeof rx_buf,RECEIVE_TIMEOUT)) { return
1; }

    /* Bluetooth init */
	err = bt_enable(NULL);
	if (err) {
		printk("Bluetooth init failed (err %d)\n", err);
		return 0;
	}
	printk("Bluetooth initialized\n");
```

```
      /* Button0 interrupt configurations */
      err = gpio_pin_interrupt_configure_dt(&button0, GPIO_INT_EDGE_TO_ACTIVE );
    gpio_init_callback(&button_cb_data, button_pressed, BIT(button0.pin));
      gpio_add_callback(button0.port, &button_cb_data);

      /* Button1 interrupt configurations */
      err = gpio_pin_interrupt_configure_dt(&button1, GPIO_INT_EDGE_TO_ACTIVE );
    gpio_init_callback(&button_cb_data1, button_pressed_choose,
BIT(button1.pin));
      gpio_add_callback(button1.port, &button_cb_data1);

      /* Button2 interrupt configurations */
      err = gpio_pin_interrupt_configure_dt(&button2, GPIO_INT_EDGE_TO_ACTIVE );
    gpio_init_callback(&button_cb_data2, button_broadcast, BIT(button2.pin));

      gpio_add_callback(button2.port, &button_cb_data2);

      /* All inits are done, start system */
      system_state == STATE_THINKING;


      while (1) {
        k_msleep(1000);
      }

      return 0;
}

K_THREAD_DEFINE(thread0_scanner, STACKSIZE, thread0, NULL, NULL, NULL,
            THREAD0_PRIORITY, 0, 0);
K_THREAD_DEFINE(thread1_button1, STACKSIZE, thread1, NULL, NULL, NULL,
            THREAD0_PRIORITY, 0, 0);
```