Raymond & Beverly Sackler
Faculty of Exact Sciences
Tel Aviv University

Keyboard-In-The-Middle

Sharon L.

Mattan S.

Hila B.

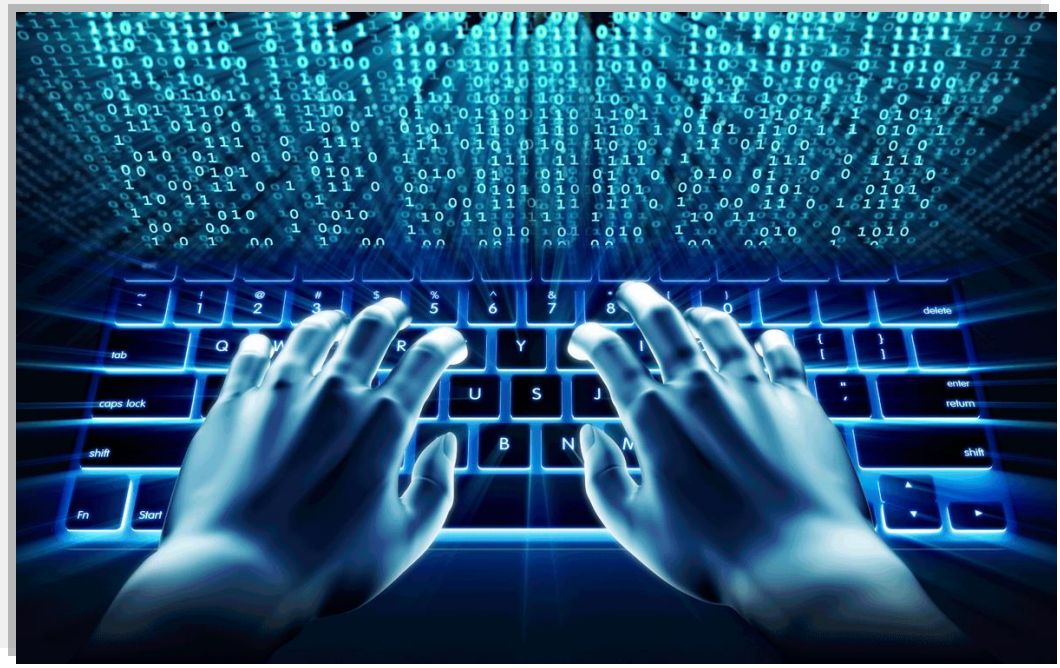
Course

Advanced Computer Systems

Project Supervisor

Prof. Sivan Toledo

School of Computer Science, Tel-Aviv University



Motivation

Today's wireless keyboards are connected to central devices using Bluetooth® Low Energy protocol with a profile called HID (Human Interface Device) over GATT (Generic Attribute).

Most of the keyboards encrypt their notification to eliminate the possibility to sniff the data transferred as it can be sensitive (password and ext..).

Our motivation is to establish MITM attack by impersonating keyboard and control all traffic in between. The project is a POC of utilizing vulnerability of adding the human factor into the equation before the pairing process occurred.

Potentially this project can be extended in multiple ways:

- Intercepting HID packets sent from the HID device to the HID Host.
- Data collection and monitoring of end users by saving it to external memory card.
- Injecting \ controlling user computers.
- Modify packets in real time.

Components

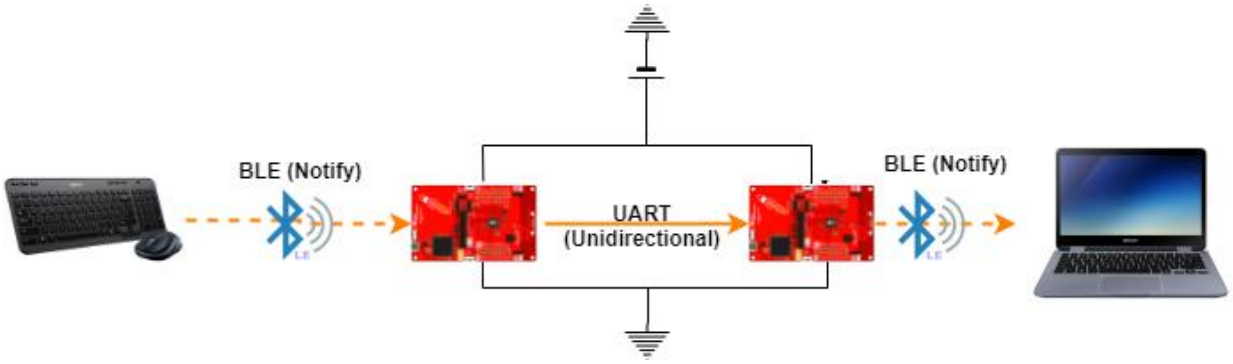
Project components:

- 2 Texas Instruments CC-2650 LP evaluation boards.
- 2 jumper cables.

Debug components:

- 1 Texas Instruments CC-2650 LP evaluation board (imitating BLE keyboard).
- 1 computer (for firmware loading and power supply).
- 1 oscilloscope and probes (for real-time UART debugging).

Block Diagram and Data Flow



Data Flow:

1. Central Device – HID Host listed to advertisements, searches for HID Profile devices and connects to one while sending its name to the Peripheral via UART.
2. Peripheral Device – HID Device gets the advertisement name and starts advertising while waiting to connection from a computer.
3. After connection has been established, the Central catches each HID Report notification sent and passes the keycodes over UART to the Peripheral which creates new HID Reports and sends them to the computer.

Live Demo

A Live demo presentation can be found in the following link:

https://drive.google.com/file/d/13XrjnG8WzOPZ_VBScC5rJdkH7CERXxDW/view

Development Process

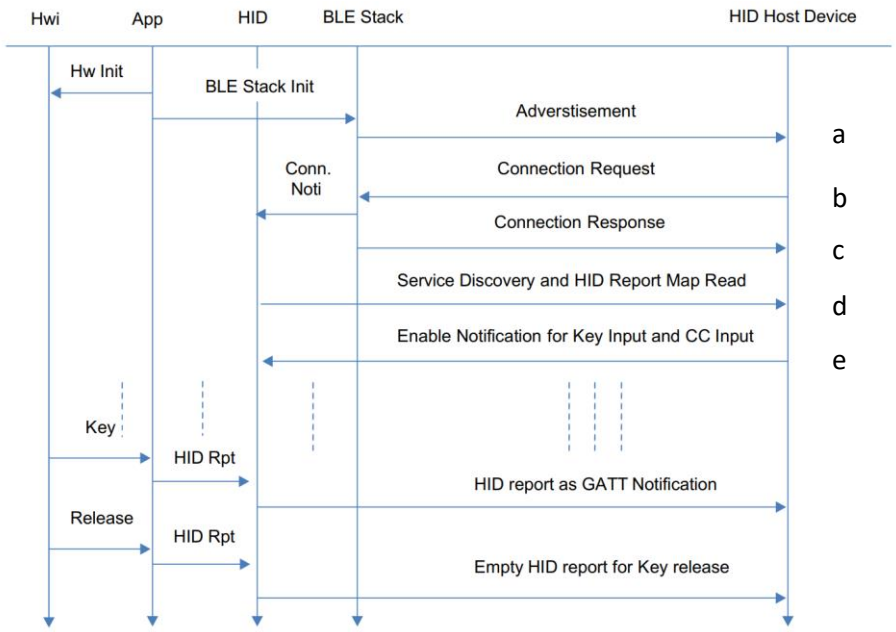
The project development process was divided into three parts:

- HID Host profile on a Central BLE device.
- HID Device profile on a Peripheral BLE device.
- UART communication between the devices.

1. HID Host profile on a Central BLE device

We used a Simple Central Project example as a starting point and extended it to support HID Host functionalities needed to generate the attack.

The connection process to a HID device over BLE is presented in the following chart:



a. Advertisements by the BLE peripheral - HID device includes his name and primary services.
b. Connection request by the BLE Central – HID host to the peripheral device.
c. Connection acknowledge response.
d. Service discovery request and obtaining the handle to each service/descriptor/characteristic.
e. Enable notification request to the desired handle.

After a basic connection between the peripheral and central was established (stages ‘a’ to ‘c’), we want to enable notifications in the HID Device by sending a specific command to the correct handle (stages ‘d’ and ‘e’).

For this action we need first to find the handle for the 'Client Characteristic Configuration' or 'CCC' descriptor which controls the Notification or Indication in the BLE protocol by these steps:

a. Discovering primary service by UUID:

   We want to discover the start and end handles for the primary HID service 0x1812 by sending this value as a 'ATT find by type value' request - 0x06 and waiting for a 'ATT find by type value' response - 0x08 which will expose these handles.

b. Discovering all characteristics descriptors within this service:

   We do so by sending 'ATT Find Information' Request – 0x04 with the start and stop handles and waiting for 'ATT Find Information' Responses – 0x05, this request generates many response and we need to catch them all. We stop when we get 'ATT Find Information' Response with status 'Procedure is completed' – 0x1A in its header.

c. Filtering only the CCC descriptors:

   We want to filter and save only the 'CCC' descriptor – this are the notification characteristics that send the keyboard data. For each 'ATT Find Information' Response we check the UUID of the descriptor and only search for 'CCC' – 0x2902. the HID service contains several CCC descriptors and we need to save the all.

d. Enabling Notification:

   For each 'CCC' descriptor we create delayed event in the BLE Central environments. The request is delayed in order to avoid missing discovery responses and complete the discovery and the connection process. The 'CCC' descriptor defines its characteristics 16 bits value as follows:

   - 0x00: Indication and notification disabled.
   - 0x01: Notification Enabled.
   - 0x02: Indication Enabled.

   For each 'CCC' descriptor handle we saved, we send a 'GATT Write with no response' request – 0x1A with value of 0x01 in order to enable notification.

After Enabling the notification to the correct characteristics, we can catch GATT Notification messages – a message with method of 0x1b. these messages are HID Reports:

modifier	Reserved	Key code 0	Key code 1	Key code 2	Key code 3	Key code 4	Key code 5

In the current implementation we only send 'Key Code 0' but it can be easily extended to send a full 8-bytes report over the UART.

2. HID Device profile on a Peripheral BLE device

We used the HidKbd project example by TI for the CC-2650 LP device and the project implements HID keyboard device based on the Simple Peripheral project example.

To be recognized as an HID, a device must implement the HID-over-GATT Profile, which means at least the following services:

- HID.
- Battery.
- Device information.

HID service:

All the structure formats described in HID are used in HID-over-GATT.

- Report Map: USB HID Report Descriptor.
- Report Reference Characteristic Descriptor: setting a report characteristic's metadata contains type (Input/Output/Feature) and ID of a report.

The HID Service defines the following characteristics:

- Protocol Mode: the default is Report mode, but you can change that to Boot mode.
- Report Map: the HID Report descriptor, defining the possible format for Input/Output/Feature reports.
- Report: a characteristic used as a vehicle for HID reports. HID Over GATT defines be one characteristic per report described in the Report Map which will include Boot Keyboard Input Report.
- Boot Keyboard Output Report.
- Boot Mouse Input Report.
- HID Information: HID version, localization and some capability flags.
- HID Control Point: inform the device that the host is entering or leaving suspend state.

HID over GATT uses input reports that sent using notifications. They can also be read by the host and contains the keycodes sent from the keyboard to the host

Battery and Device Information host:

Basic services containing information about the device and a battery indication.

This device operation divided into two steps:

1. On startup, waits for a UART packet that contains the name of the keyboard we want to imitate.
2. After the name was received we start the HID service with the new name and wait for a connection from a computer.

The device will receive from the central from now on packet over UART that contains keycodes needed to be sent.

3. UART Communication between the devices

We used example from the EchoUart project and changed the behavior to our needs.

While developing the UART Communication we had problems maintaining the BLE Connection of the Peripheral and Central device with the keyboard and laptop, after consulting with the TI support forum we decided to use the UART Callback interface and maintain the minimal amount of data manipulation in those CB before storing the packet in a queue and returning to the main BLE task.

Our UART Interface has 2 working modes:

- Transfer stolen Keyboard name.
- Transfer keyboard keycode.

a. Transfer stolen Keyboard name

In order to transfer a keyboard name from the central device to the peripheral we defined a TLV protocol to support string with different sizes and to avoid recognizing noise on the line ad valid data.

The packet structure is:

<barker::byte><size::byte><keyboard_name::char_array>

The barker was set to '&' and basically here to protect us from recognizing other data as valid packets. We have seen that while the central device configuring the UART lines, the Peripheral recognized 'junk' as valid data and the barker protects us from it.

This packet is send by the central device and the peripheral uses the incoming data to configure its name.

Central UART configuration: The UART configured as Callback in order to work alongside the BLE stack and the callback function is empty as we don't need to do nothing after the data was sent.

Peripheral UART configuration: The UART configured as Blocking because the HID Services are not enabled, and the device waits for this data to start to work.

b. Transfer keyboard keycode

After the Peripheral name was configured, both devices move to handle keycodes transfer.

This protocol is simply passing single keycode over the UART interface. Each keycode the Central gets from the keyboard passed to the peripheral and transmitted to the computer. In this project we only support passing keycode and not passing modifiers, but this can be easily extended by using the same protocol defined in the section above and transferring full hid report.

Central UART configuration: The UART configured as Callback in order to work alongside the BLE stack and the callback function is empty as we don't need to do nothing after the data was sent.

Peripheral UART configuration: The UART configured as Callback in order to work alongside the BLE stack and the callback function only generate a report and enqueues it to the application main queue.

Challenges

1. Multiple boards environment

   This project contained developing and testing multiple boards simultaneously. The TI environment and example project all relay on the same files and imported to all of the projects. Changing one file interrupted to the other project which forced us to try to maintain two SDK occurrences and in some case work on different computers. More over the serial numbers of the devices collided and forced us to change the in order to work on the same computer:

   http://processors.wiki.ti.com/index.php/XDS110#Finding_and_updating_the_serial_number

2. Porting projects from CC-2650 to CC-1350

   At the beginning we tried intensely to port the example project of the CC-2650 to the environment of the CC-1350 in order to work with our own cards. This was a major time waste all our attempts fail and after trying to solve this issue with the TI support team on their forum the we realized that nobody there has done it but they claim that it can be done. One of the major setbacks here was the different BLE SDK used by the 2 projects – a big chunk of the code has been changed and don't match the CC-2650 old projects.

3. Trying to use MultiRole example project

   After turning to use the CC-2650 we tried to avoid using 2 boards for this project. We tried to port the HidKbd project into the MultiRole project that support maintaining up to 8 connections as Central or Peripheral. After trying there we got to the conclusions that it's not going to fit into the code space of the CC-2650, we also tried to check this with the TI support and they think so too.

4. Central HID Host profile

   In our project we didn't implemented a full HID Host profile and only used the basic needs to catch the notifications. TI does not have a HID Host profile example, and this had to be done by us. To understand how to do so, we had to learn the example projects of SPP and Bi-directional audio that and to build the data flow for this action and understanding what is the proper data configuration needed to enable the notifications. The TI forum didn't help us so much in this subject, but we posted the solution there for future help.

5. UART serial communication

   Enabling UART on going communication along with BLE communication was a great challenge. The first attempt was to create a new task that will handle the UART_read() functionality and send HID report over BLE. This attempt had failed from few reasons. While the task was acquiring the OS resources the BLE connection that was handled by another task froze and as a result got disconnected. The second reason was that we didn't ensure ground voltage coordination on both boards- the Central and the peripheral.
   The second attempt was configuring the UART read functionality to be handled by a callback other than new task. The callback function woke up as soon as data arrived the UART line. We have read the data and tried to send HID report over BLE. This attempt failed too. After deep analyzing and forum consultations we understood that callback can't perform heavy tasks as sending the HID report over BLE. In order to avoid performing the send operation in the callback we enqueue the packet in the outgoing HID buffer and the sending now is handled by another function.

Additional Links

1. https://docs.mbed.com/docs/ble-hid/en/latest/api/md_doc_HIDService.html
2. https://e2e.ti.com/support/wireless_connectivity/bluetooth_low_energy/
3. http://processors.wiki.ti.com/index.php/XDS110#Finding_and_updating_the_serial_number