Imagine this: you have just bought a swanky new car with a Bluetooth Car Kit and as you are learning to experience the new gadget, something bizarre happens - your serene drive is suddenly interrupted by a ghastly voice emanating from your car speakers. Baffling? Not if you know about “Car Whisperer”.

Security has been an important matter for the Bluetooth Special Interest Group (SIG). Though Bluetooth is inherently secure and the specifications provide all the necessary nuts and bolts for an implementer to put together a secure Bluetooth product, a chain is only as strong as its weakest link. A Bluetooth product is secure only if the implementer uses all the security features effectively. Incorrect implementation can make a Bluetooth device unsafe and render a user skeptic about its security.

The initial section of this article outlines the security features of individual Bluetooth components, and the subsequent section drills-down into the application component explaining the various types of Bluetooth security breaches and recommended solutions to thwart them from the product application.

Security of individual modules

Typically, Bluetooth components in a product are integrated rather than implemented from scratch. The host-based solution (also referred as two-chip solution), is common in many products, as shown in the Figure below:

Figure 1: Host Stack Architecture. To build a completely secure product, the individual components integrated must conform to appropriate security requirements. The following table provides information about the security features & responsibilities of individual components:

<table>
<thead>
<tr>
<th>Component</th>
<th>Security Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMP, Baseband</td>
<td>• Supports all security related PDUs and sequences</td>
</tr>
<tr>
<td>Upper Layer Stack and Profile</td>
<td>• Supports security modes 2 and 3 and selection of security mode</td>
</tr>
<tr>
<td></td>
<td>• Supports setting page scan modes</td>
</tr>
<tr>
<td></td>
<td>• Supports variable length PINs</td>
</tr>
<tr>
<td></td>
<td>• Supports a well-designed Security Manager with interfaces to set devices as trusted /untrusted</td>
</tr>
<tr>
<td></td>
<td>• Supports enabling/disabling encrypted links</td>
</tr>
<tr>
<td>Application</td>
<td>• Appropriate selection of security features based on product requirements</td>
</tr>
<tr>
<td></td>
<td>• Implementation of product use cases in a secure manner</td>
</tr>
<tr>
<td></td>
<td>• Provides balanced user experience, without compromising on security</td>
</tr>
</tbody>
</table>

Secure usage of the product is the responsibility of the user, however, a good implementation should aid in preventing the user from unintentionally using the product in an insecure manner (for example, features like suitable display of warnings/tones when the user is about to do something potentially unsafe.) Let us examine how the product application can be implemented in a secure manner through a product example.

Security considerations in a product application:

We will consider the example of a Hands-free Car Kit (HFCK); the security attacks it may be prone to; and how these can be prevented in the implementation. HFCKs support features such as call handling and music streaming. The profiles typically required to realize these features are:

- HF (Hands-free Profile)/HS (Headset Profile)
- OPP (Object Push Profile) and/or PBAP
- A2DP (Advanced Audio Distribution Profile) and AVRCP (Audio Video Remote Control Profile)

Some of the security attacks HFCKs may face are:

(i) Eavesdropping or “Car Whisperer”
(ii) “Bluejacking”
(iii) Stealing phonebook data

Let us consider, that we are dealing with a sophisticated hacker, who has an air sniffer, PC-based tools, a directional antenna to increase the range, and can also follow around the car at high speeds!

The minimum information that the hacker needs is to “know” the device to be hacked is the BD_ADDR of HFCK. One of the scenarios for the hacker to find the BD_ADDR is when the HFCK is in a discoverable mode. All that the hacker needs to do is to execute an inquiry to find the target device. From the device names, the hacker can guess the target device rather easily. Keeping a device in discoverable mode at all times has been one of the mistakes committed by the early implementations. Fortunately, the new implementations take care to switch on the discoverable mode only when required. For a product like HFCK, the following guidelines would apply:

- HFCK must not be discoverable on startup
- HFCK must be put into discoverable mode (“pairable” mode as it may be referred to on UI) by the user to pair with other phone for the first time
- Pairing must be performed in secure environments since the data captured during the pairing procedure can potentially be used to compute the link key. The user must be alerted through a suitable warning on screen and audio beeps that pairing must be attempted only in safe environment, such as the user’s home/garage
- After the pairing is successful, HFCK must come out of discoverable mode and become connectable, so that only a “known” device can connect
- A timeout period can be implemented, which means that if the pairing is not successful within the given time duration, the HFCK automatically comes out of discoverable mode. User must put HFCK into pairable mode again for it to be paired
- Number of retries for pairing option can also be provided

Let us consider another extreme case of a Bluetooth security breach. In spite of taking care of the above guidelines, the...
The product use cases can, many times, be secured. By using Security Mode 3 or Security Mode 2 with authentication mandated for OPP Service. In either of these cases, authentication is done for each contact push. This requires the sender of the contact to enter the PIN the first time and for subsequent times the link key authentication is done as depicted in Figure 3. This option takes more time and may not have a positive impact on the user’s experience.

Another option is to give a thought to dial-out calls directly from the HFCK while driving, without having to reach for the mobile phone. The effect of Bluejacking is more of a nuisance, than a serious security lapse.

Let us assume that the hacker has somehow acquired the BD_ADDR of the HFCK. The hacker can then simply use a PC-based tool to send a contact over OPP to the HFCK’s BD_ADDR. This problem can be prevented using one of these options:

- By using Security Mode 3 or Security Mode 2 with authentication mandated for OPP Service. In either of these cases, authentication is done for each contact push. This requires the sender of the contact to enter the PIN the first time and for subsequent times the link key authentication is done as depicted in Figure 3. This option takes more time and may not have a positive impact on the user’s experience.

• Another option is to give a thought about application use case i.e., the user would only push a contact from their mobile phone, which would also be used for Hands Free (HF). Thus, the mobile should already have been paired for using HF. Hence, the application would need to allow contact push from only a device paired for HF and simply reject OPP connections for all other devices.

### (ii) Bluejacking:

Would we be seeing the famed “Bluejacking” [http://en.wikipedia.org/wiki/Bluejacking] now on HFCKs? This attack involves sending an unwanted message, masqueraded as a contact file, through use of OPP. HFCKs support OPP to receive phonebook contacts from the mobile phone, which are stored locally for the user to dial-out calls directly from the HFCK while driving, without having to reach for the mobile phone. The effect of Bluejacking is more of a nuisance, than a serious security lapse.

Let us assume that the hacker has somehow acquired the BD_ADDR of the HFCK. The hacker can then simply use a PC-based tool to send a contact over OPP to the HFCK’s BD_ADDR. This problem can be prevented using one of these options:

- By using Security Mode 3 or Security Mode 2 with authentication mandated for OPP Service. In either of these cases, authentication is done for each contact push. This requires the sender of the contact to enter the PIN the first time and for subsequent times the link key authentication is done as depicted in Figure 3. This option takes more time and may not have a positive impact on the user’s experience.

Another option is to give a thought about application use case i.e., the user would only push a contact from their mobile phone, which would also be used for Hands Free (HF). Thus, the mobile should already have been paired for using HF. Hence, the application would need to allow contact push from only a device paired for HF and simply reject OPP connections for all other devices.

### (iii) Stealing Phonebook data:

HFCKs have a locally stored phonebook downloaded from the user’s mobile phone. Can a hacker access this sensitive information? Let us assume that the hacker has somehow obtained HFCK’s BD_ADDR and can simply issue an OPP PULL request. The following considerations can help an implementer avoid this issue:

- Is the Phonebook PULL functionality required to be supported by HFCK? If it is not really required, can we simply reject all such PULL requests?
- If this feature is required (let us say, the user has lost the data on the mobile phone and wants to retrieve the data available on the HFCK back to the phone), can it be made a user-initiated PUSH instead of PULL?

To summarize, HFCK can be guarded against security attacks by following some rules during product implementation:

- HFCK should be put into discoverable mode only through user action when required
- Instead of “Standard” PINs, randomly generated PINs (dissimilar for each pairing session) should be used
- Consider options such as a user-specified PIN or a unique PIN for each device of the same make
- The product use cases can, many times, give insights for a secure implementation

For our discussion, we have considered Car Kit as an example. The guidelines provided are also applicable for other product types (laptops, headsets, mobile phones) as well.

Girish Managoli is a Technical Manager at MindTree Consulting – a global IT consulting and R&D services organization providing product realization services to its customers. In his position at MindTree, Girish is responsible for adaptation of MindTree’s product ready Bluetooth suite (EtherMind) to customer platforms and products. Girish holds a Bachelor’s degree in electronics engineering from University of Mysore, India.