[BGScript]: whitelist_peripheral - BLE peripheral with pairing/bonding and whitelisting

Jeff Rowberg
posted this on December 27, 2013, 00:50

Project Name:	whitelist_peripheral
Description:	Demonstrates **whitelisting** within the context of a BLE peripheral. Whitelisting is the technique used to restrict which devices are allowed to connect to your device. If a remote device is not on the whitelist, then it is not allowed to connect. **Whitelisting with iOS devices will not work due to a limitation in the way the internal chipset in the BLE modules implements stack-level whitelisting functionality. See the "Additional Notes" section below for detail.** This example has the following features: • **Bonded addresses and address types** are stored in PS keys (retained across power-cycles/resets, but not across reflashes) • **Unfiltered advertising** is used automatically when no bonding entries are present (i.e. anyone can connect) • **Filtered advertising** is used automatically when 1 or more bonding entries are present • **Pressing the P0_0 button** on the devkit (or pulsing that pin high) will enter "pairing mode", i.e. close active connections and start unfiltered advertising, even if bonding entries do exist • **Pressing the P0_1 button** on the devkit (or pulsing that pin high) will erase all bonding/whitelisting entries, close active connections, and start unfiltered advertising • **UART1/Alt1 @ 115200** without flow control is used for debug output to show what is happening at each step (see screenshot below) • **The GATT structure has a single special "authenticated read" characteristic** which you can use to test whether encryption is working properly (if you can read it from LightBlue on iOS, then it's working)
Instructions:	1. Unzip **whitelist_peripheral.zip**. 2. Compile and flash **project112.bgproj** or **project113.bgproj** with BLE Update utility. 3. Connect a serial terminal such as Realterm or PuTTY to the UART1/Alt1 port @ 115200 (optional). 4. Connect to the device using LightBlue or a similar app on iOS, or with the BLED112 and BLEGUI on Windows. 5. The peripheral will automatically request pairing/bonding with the master if the master does not. The "just works" pairing mode is used, so no PIN entry is required.

	6. After connecting/bonding with one device, only that device will be able to connect in the future, UNLESS either #7 or #8 below are done: 7. Press the P0_0 button to enter "pairing mode" (allow connections from anyone). 8. Press the P0_1 button to clear all bonds and whitelist entries and start fresh.
Notable Functions:	• **Command**: sm_set_bondable_mode • **Command**: sm_encrypt_start • **Command**: sm_get_bonds • **Command**: system_whitelist_append • **Command**: system_whitelist_clear • **Command**: gap_set_filtering • **Command**: flash_ps_save • **Command**: flash_ps_load • **Event**: sm_bond_status • **Event**: sm_bonding_fail
Safe to Flash:	BLE112, DKBLE112, BLE113, DKBLE113
Unsafe to Flash:	BLED112 (no ability to return to DFU mode, GPIOs unavailable anyway)
Additional Notes:	## Whitelisting guidelines The whitelist on the module can only be modified **while you are not already advertising, scanning, or connected**. This example implements whitelist modifications correctly in this regard. Here are the basic steps: 1. At boot time, load any stored MAC addresses/types from PS keys and add to whitelist 2. Set filtering policy and start advertising 3. Upon connection + bonding with new device (in **sm_bond_status**), store new MAC/type in PS key 4. Upon disconnection from new device, add MAC/type to whitelist 5. Resume advertising ONLY AFTER step #4, or else that address will not be properly whitelisted You can stop advertising and set non-filtering policies at any time to allow connections from any device, even if the whitelist has entries added to it. This example project does this when you use the P0_0 button on the DKBLE11x devkit. ## Lost bond data Bonding data and the manually stored MAC address/type values are saved in the public store (PS) keys on the module itself. This information is retained across resets and power cycles, but **it will be lost if you reflash the device with new firmware**. This means that if you bond with another device, then make a change to the firmware and reflash it, the bonding information will be missing on only one side of the connection.

This can cause unexpected behavior if you are not aware that it has happened. **If you reflash the firmware or otherwise clear bonding info after bonding with an iOS device, you MUST go into the iOS device's Bluetooth Settings area and "forget" the bonded device before attempting to re-bond.**

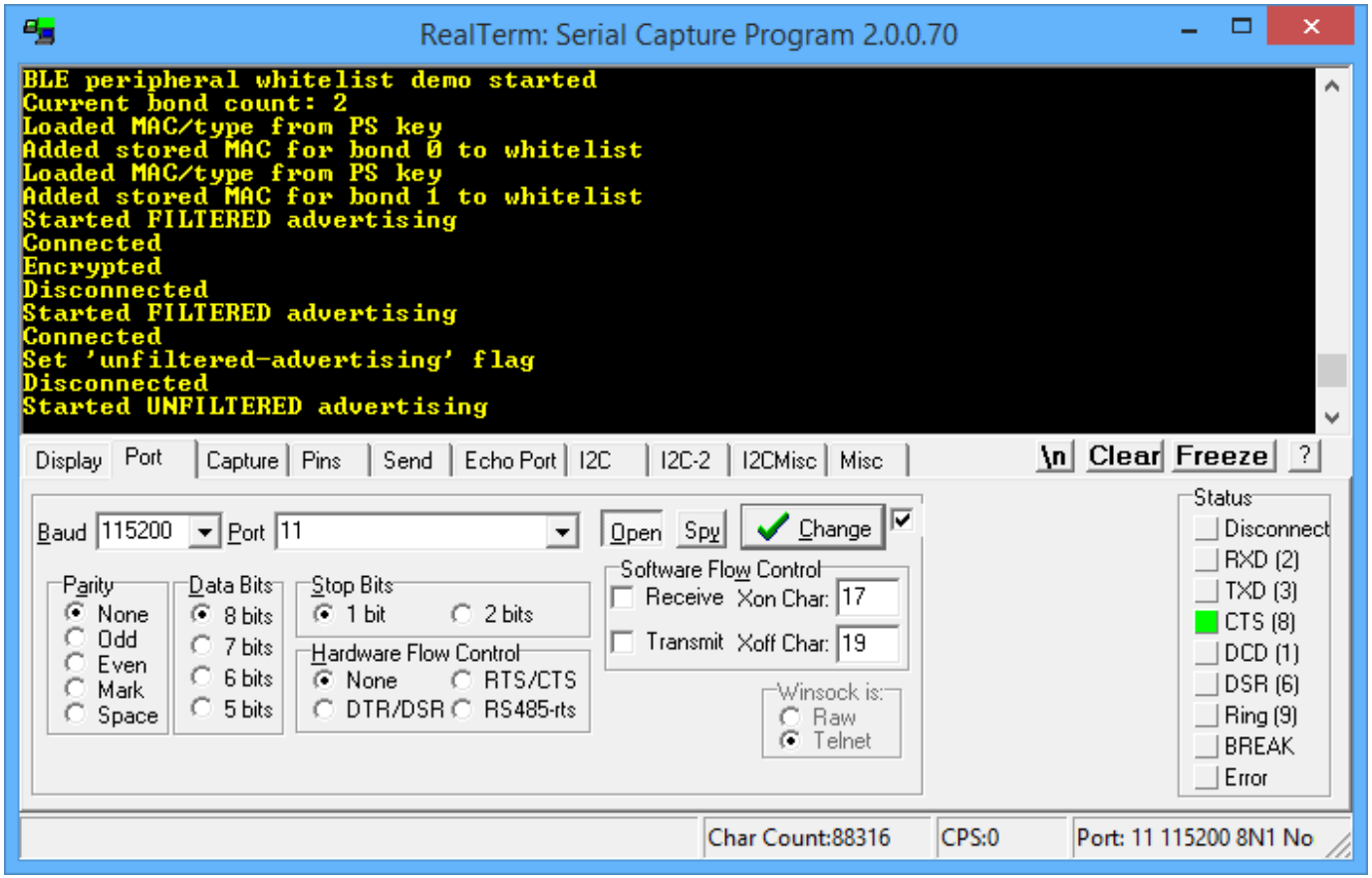
iOS and random/private addressing limitations

The CC254x chipset used in the BLE(D)11x devices has a hardware limitation which prevents a whitelisted private/random address from continuing to match after the remote device changes its address, EVEN IF THE DEVICES ARE BONDED. Whitelisting only matches the address and address type bit pattern transferred over the air, and does not make use of the IRK (identity-resolving key) which would allow the whitelisting filter to "follow" a changing private address.

This means that devices which employ the private addressing scheme with a rapid address update interval (most notably **ANY iOS DEVICE**) cannot be whitelisted using the built-in stack method demonstrated in this example. An alternative method would be to use bonding (as this example does), but write in similar application-level functionality which does not use the stack's whitelisting features but instead simply disconnects immediately (using "**connection_disconnect**") if a device connects without a valid bond handle. This is effectively what the stack's whitelisting functionality does, just a little bit less efficiently.

TO REITERATE: this example will not work with an iPhone/iPad for more than about 15 minutes. As soon as the iOS device updates its private address--which occurs on an interval as well as any time the Bluetooth subsystem is stopped/started, e.g. cycling flight mode--then whitelisting the original address will prevent the device from connecting again.

The "**address_type**" argument to the "**system_whitelist_append**" command is still useful for whitelisting if the remote device uses static addresses, i.e. an address which is random to the device but does not change often. More info on this is available in the Bluetooth 4.0 Core Spec Vol 3, Part C 10.8.



whitelist_peripheral.zip

3 people found this useful. - **Me too!**