**Test Procedures for BLE Manual Testing**

**BLE device scanning techniques** - Using linux utilities we can scan near by Bluetooth devices.

* Packages to be installed in our attacking system:-

sudo apt-get install Bluetooth bluez libbluetooth-dev libudev-dev

systemctl enable Bluetooth.service

service Bluetooth start

sudo hciconfig hci0 reset

sudo hciconfig hci0 up

hciconfig info

* **For enabling BLE scan:**

Insert the Bluetooth dongle and make sure that the dongle supports ble scanning

hciconfig

journalctl -f # to check the dongle is properly inserted and the system could reached the target or not

lsusb # to verify bus is assigned to the Bluetooth adapter

hciconfig hci0 version

btmgmt info # to know the details of hci0 interface

btmgmt le on # to set the interface ready for le scan

btmgmt info

Again make the adapter up

hciconfig hci0 up

* **For BLE Scanning**

hcitool lescan # scan le devices near by

spdbrowse tool #for service and characteristic enumeration

**BLE – An introduction**

* No channel hopping which facilitates sniffing and MITM
* The essence of BLE hacking can be described as follows
  + Device discovery
  + Reading its services and characteristics
  + Detection writable characteristics
  + Mapping characteristics id’s to its functions
  + Change characteristic values

The attacks can be categorized as three parts:-

* Enumeration
* MITM
* Flooding
* Fuzzing

**Enumeration -** Tools used for enumeration is bettercap and linux Bluetooth utilities.

**Bettercap-** It is a powerful, easily extensible and portable framework written in go which aims to offer to security researches, red teamers and reverse engineers an easy to use, all in one solution with all the features they might possibly need for performing attacks on BLE, WiFi,wireless HID and Ethernet devices

**We are using it for BLE scanning, characteristics enumeration, reading and writing**

bettercap # scans in each second

ble.recon on #to start ble device discovery

ble.recon off # to stop discovery

ble.clear # to clear all ble devices

ble.show # to show discovered ble devices

ble.enum MAC # to enumerate service characteristics and data value for the given ble

ble.write MAC UUID HEX\_DATA

**Alternate method : By using hcitool and gatttool for performing the same action (enumeration)**

sudo hcitool lescan

sudo hcitool leinfo <macid>

sudo hcitool lecc <macid>

Connection handle 64

sudo hcitool ledc <handle value>

**We can use gatttool for enumeration of services and characterestics**

gattool -I

connect <mac id >

char-read-uuid <uuid> alternatively char-read-hnd <handlevalue>

char-write-cmd <handlevalue> <value> or Char-write-req <handlevalue>

**Note: Inorder to write to a specific handle, we need to know which one is a write handle. For this, we can go for a hit try method and try reading all the handles one by one until we encounter a read error. A read error means the specific handle is a write handle**.

**MITM :-** The tools used for carry out MITM is nrf connect, wireshark and Btlejuice.

**Btlejuice Framework:-** The BLE device is a smart bulb or any smart devices we can do mitm with Btlejuice framework.It requires two systems:-

|  |  |
| --- | --- |
| **Kali Host** | **Kali VM** |
| * NPM 6.14.0 and node 8.9.0 | * NPM 6.14.0 and node 8.9.0 |
| * Btlejuice installed (npm install -g btlejuice and npm install -g btlejuice-bindings) * sudo service Bluetooth stop * sudo hciconfig hci0 up * Btlejuice -u <vmip> -w * Browse to <http://localhost:8080> to interact with the bulb to do mitm | * Btlejuice installed (npm install -g btlejuice and npm install -g btlejuice-bindings) * sudo service Bluetooth start * Btlejuice-proxy |

Btlejuice can act as a proxy through which the communication between the smart phone and smart bulb takes place. We can do the following things with Btlejuice framework

* Interception
* On the go data modification
* Replay attack
* Export the data to json file

**HACK BLE communicating with mobile phones**

Tools used :- nrf connect app to connect it and to write characteristic values. The app indicates which services or characteristics can be writable. To get the required value enable Bluetooth HCI snoop log in mobile by doing the following steps.

adb pull /sdcard/btsnoop\_hci.log

adb pull /sdcard/Android/data/btsnoop\_hci.log

Open in wireshark and filter with btatt.opcode.method==0x12 or Bluetooth.addr==== <mac id >

Get the writable value and send it with nrf connect app.

**DoS Attack/ Flooding Attack**

**L2ping Attack :-** Here we can do flooding of l2ping packets to the target ble device and check the availability of the device with Grapo tool.

Target device:- JBL Bluetooth speaker

**Script link:-** [**https://github.com/crypt0b0y/BLUETOOTH-DOS-ATTACK-SCRIPT**](https://github.com/crypt0b0y/BLUETOOTH-DOS-ATTACK-SCRIPT)

Input the target address, packet size and number of threads to the script and run as python3 Bluetooth-DOS-Attack.py

**Steps:**

sudo service Bluetooth start

hcitool lescan

python3 Bluetooth-DOS-Attack.py

<target addr>

<packet\_size>

<number of threads>

Run Grapo tool in parallel to check the availability.

Continuously Monitoring (hcitool,l2ping)

**Fuzzing**

**L2cap Fuzzing Attack – Bluetooth Stack smasher Tool (predsac , documentation)**

Installed and verified the tool is working fine. Tried with the JBL speaker but it didn’t make any significant attack on speaker.

Usage :- ./bss -M 0 -m 13 -s 10 <mac addr>

In this tool we can replay the buggy packet which cause the potential crash.

(prerequisite, explain it in detail)

**Currently working on:-**

**Connection establishment in BLE**

Advertisement packet

Scan response request

Response data

connected

Peripheral device Central Device

**Rogue Attack** :- Here by spoofing the ble mac address then by impersonate as a trusted ble device which further communicate with the master (central device).