CSAW 2018

Embedded Security Challenge

Final report

Eléonore Carpentier - *INSA Centre Val de Loire*
Corentin Thomasset - *Polytechnique Montréal*

Team name : The Maple Cookie Army
Team advisor: Jeremy Briffaut - *INSA CVL*

I- Introduction

IoT devices are invading our lives. From homes to working places they are everywhere. This massive adoption brings some security issues as those devices are designed to lower the time to market and therefore neglect most of security recommendations and best practices. This is even more problematic when they are used in highly secured environment containing sensitive data. Their presence creates security holes and threatens network security as their security level is often below requirements.

In this context, we'll present a technique to exfiltrate data from a highly secured networks by misusing a smart bulb.

II- Scenario

Our target is a highly secured network, physically isolated from the internet. The goal is to exfiltrate sensitive data without being spotted and with the smallest fingerprint as possible. Based on recent intels, the site has a dedicated security operation center operating high end firewalls, IDS and IPS and any attempts to access, intercept or modify physical network packets with an unauthorized device would be seamlessly spotted. Moreover, employees are subject to very strict security measures: It is strictly forbidden to bring in devices from outside the company and every employee goes through a security screening when arriving and leaving the protected area. The screening procedure is similar to TSA controls in airports and it would be too dangerous to try to bring a device inside or to leave with an USB stick. Thus, solutions need to fully work on on-site devices and are required to communicate data with our ground team outside the building. According to our sources each employee has at his own disposition a laptop and a professional cellphone. However, they are not granted any admin right and cannot install third party software without getting through a long and fastidious process. Recently some Bluetooth smart bulbs have also been deployed and employees are allowed to control them to adjust the light for better working conditions. Smart bulbs can provide a reliable medium to bridge the air gapped network, but it is unclear if Bluetooth wireless connection are also under surveillance. As failure is not an option, we will consider wireless connection under the same surveillance as the classical network.

III- Initial considerations, context and assumptions:

In the previous scenario, we consider the worst-case scenario where attack vectors are very limited. We have designed our attack to work under the following circumstances:

- The physical network is isolated and not connected to the internet: No third-party software/library can be downloaded.
- Employees have no admin rights on workstations & laptops. No third-party software installation is possible. We consider an out of box set-up where only pre-installed work software are available to use.
- Wireless networks (Bluetooth) are monitored and suspect connections involving an untrusted device are reported to the Security Operation Center. Thus, Magic Blue Bluetooth smart bulbs can be controlled by employees with trusted devices only (Workstations, laptops and cellphones). However, connection can be sniffed by passive devices in the emission range without being notified.
- Security screening prevent employees from bringing in or taking out devices such as USB sticks or smartphones.

IV- **Bridging the gap: Simple but deadly BLE exfiltration technique**

Wireless networks are great for usability but a nightmare in terms of security. Their range varies widely depending on the environment and it’s almost impossible to restrain them to a certain group of users as the signal can be received by any “close enough” device. In the Bluetooth Low Energy case, “close enough” is defined to be up to 100m and for the Magic Blue Bluetooth bulb, the manufacturer specifies that it can be controlled over a maximum of 25 meters. Even if walls and furniture lower drastically the emitting range, the nature of this medium allows an attacker to sneak behind a wall and sniff all unencrypted connection.

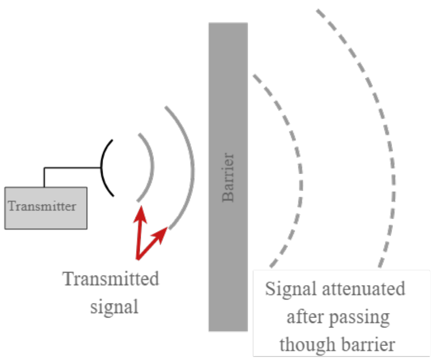
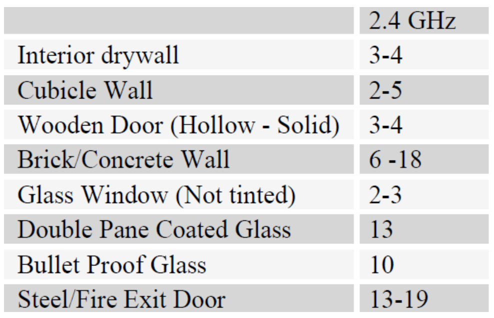


Figure 1: Signal attenuation



	2.4 GHz
Interior drywall	3-4
Cubicle Wall	2-5
Wooden Door (Hollow - Solid)	3-4
Brick/Concrete Wall	6 -18
Glass Window (Not tinted)	2-3
Double Pane Coated Glass	13
Bullet Proof Glass	10
Steel/Fire Exit Door	13-19

Figure 2: Common objects and corresponding attenuation in dB for Bluetooth Low Energy (2.4 GHz).

Moreover, Bluetooth low energy uses the same 2.4 GHz radio frequencies as WiFi. Even if the transmission range also depends on the transmitter power of emission, it’s very common to be able to sniff wifi through walls and the same principle can be applied to BLE.

Based on these observations we have designed a method to covertly exfiltrate data over IoT devices Bluetooth connections. The principle is simple : the attacker sends Bluetooth packets with a hidden data payload to the IoT device and anyone within the emission range can sniff the connection and recover the data as this traffic is unencrypted. This method prevents the attacker from being spotted by using an active device to receive the data and can be derived into two approaches:

The first approach consists in sending malformed packets to the bulb. During our experimentations, we have observed that the Magic Blue Bluetooth bulb was ignoring packets that does not conform to a specific format. This allows an attacker to send its payload directly to the bulb as the bulb won’t try to interpret it. This solution has multiple advantages : It has good transmission speeds and is invisible as the bulb won’t react to those commands. However, on the network level, despite of using a Bluetooth connection between two trusted devices, the malformed packet’s content and their frequency can be flagged as suspicious by an intrusion detection system monitoring the Bluetooth connections.

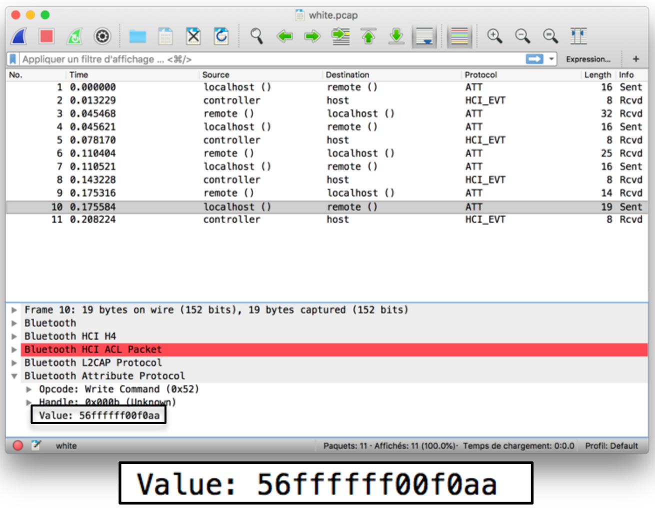
To mitigate this downside, we have developed a stealthier method on the network level. Our second approach uses steganography to hide data in the Bluetooth commands used to control the bulb. The payload is embedded in the least significant bits of each color channel in legitimate color change commands.

For example, the payload ‘Skynet’ can be encoded using 4 color change commands when using the 4 least significant bits of each color channel :

Letters	**S**	**k**	**y**	**n**	**e**	**t**
Hexa	53	6B	79	6E	65	74

	Red channel	**Green channel**	**Blue channel**
1st packet Before → After	FF → F5	FF → F3	FF → F6
2nd packet Before → After	FF → FB	FF → F7	FF → F9
3rd packet Before → After	FF → F6	FF → FE	FF → F6
4th packet Before → After	FF → F5	FF → F7	FF → F4

When using the 4 least significant bits of each color channel, the resulting bulb’s color change is unnoticeable and allows us to exfiltrate 12 bits of data per Bluetooth command while remaining unsuspicious on the network level. To be even more stealthy, the attacker can also delay each command to evade detection



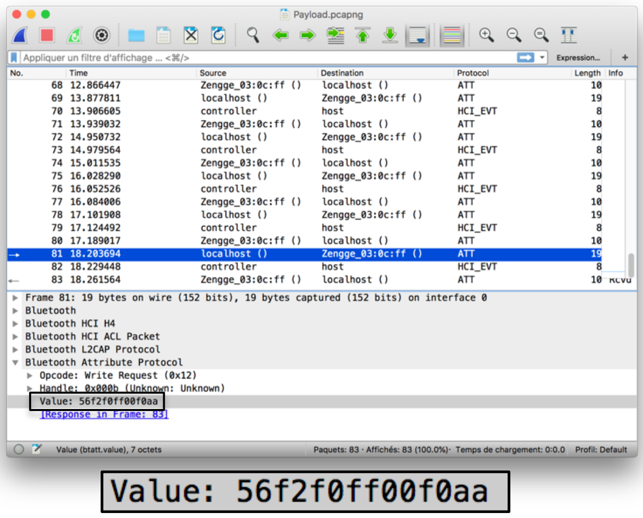
```
No.  Time      Source         Destination    Protocol  Length Info
  1 0.000000   localhost ()   remote ()      ATT           16 Sent
  2 0.013229   controller     host           HCI_EVT        8 Rcvd
  3 0.045468   remote ()      localhost ()   ATT           32 Rcvd
  4 0.045621   localhost ()   remote ()      ATT           16 Sent
  5 0.078170   controller     host           HCI_EVT        8 Rcvd
  6 0.110404   remote ()      localhost ()   ATT           25 Rcvd
  7 0.110521   localhost ()   remote ()      ATT           16 Sent
  8 0.143228   controller     host           HCI_EVT        8 Rcvd
  9 0.175316   remote ()      localhost ()   ATT           14 Rcvd
 10 0.175584   localhost ()   remote ()      ATT           19 Sent
 11 0.208224   controller     host           HCI_EVT        8 Rcvd

▶ Frame 10: 19 bytes on wire (152 bits), 19 bytes captured (152 bits)
▶ Bluetooth
▶ Bluetooth HCI H4
▶ Bluetooth HCI ACL Packet
▶ Bluetooth L2CAP Protocol
▼ Bluetooth Attribute Protocol
   ▶ Opcode: Write Command (0x52)
   ▶ Handle: 0x000b (Unknown)
     Value: 56ffffff00f0aa

white        Paquets: 11 · Affichés: 11 (100.0%)· Temps de chargement: 0:0.0   Profil: Default

Value: 56ffffff00f0aa
```

Figure 3: White color change command.



```
No.   Time       Source               Destination          Protocol  Length Info
   68 12.866447  Zengge_03:0c:ff ()   localhost ()         ATT           10
   69 13.877811  localhost ()         Zengge_03:0c:ff ()   ATT           19
   70 13.906605  controller           host                 HCI_EVT        8
   71 13.939032  Zengge_03:0c:ff ()   localhost ()         ATT           10
   72 14.950732  localhost ()         Zengge_03:0c:ff ()   ATT           19
   73 14.979564  controller           host                 HCI_EVT        8
   74 15.011535  Zengge_03:0c:ff ()   localhost ()         ATT           10
   75 16.028290  localhost ()         Zengge_03:0c:ff ()   ATT           19
   76 16.052526  controller           host                 HCI_EVT        8
   77 16.084006  Zengge_03:0c:ff ()   localhost ()         ATT           10
   78 17.101908  localhost ()         Zengge_03:0c:ff ()   ATT           19
   79 17.124492  controller           host                 HCI_EVT        8
   80 17.189017  Zengge_03:0c:ff ()   localhost ()         ATT           10
→  81 18.203694  localhost ()         Zengge_03:0c:ff ()   ATT           19
   82 18.229448  controller           host                 HCI_EVT        8
←  83 18.261564  Zengge_03:0c:ff ()   localhost ()         ATT           10 Rcvd

▶ Frame 81: 19 bytes on wire (152 bits), 19 bytes captured (152 bits) on interface 0
▶ Bluetooth
▶ Bluetooth HCI H4
▶ Bluetooth HCI ACL Packet
▶ Bluetooth L2CAP Protocol
▼ Bluetooth Attribute Protocol
   ▶ Opcode: Write Request (0x12)
   ▶ Handle: 0x000b (Unknown: Unknown)
     Value: 56f2f0ff00f0aa
     [Response in Frame: 83]

Value (btatt.value), 7 octets   Paquets: 83 · Affichés: 83 (100.0%)· Temps de chargement: 0:0.0   Profil: Default

Value: 56f2f0ff00f0aa
```

Figure 4: White color change command with data embedded. The resulting color is a slightly blueish white.

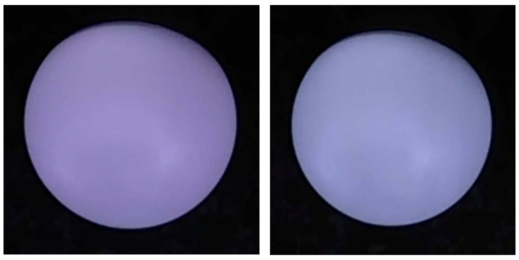


Figure 5: The bulb lightened in #FFFFFF (left) and #F2F0FF (right)

The color difference on the above picture is really subtle and it’s even harder to differentiate in real life. This is even more true in a working environment where people don’t fix the light and where a slightly color change will go unnoticed.

Finally, the data can also be encrypted during the encoding process to make sure that intercepted Bluetooth commands could not leak the payload even if the whole connection was recorded. We use a Xor cipher with a pseudorandom 256 bits key computed from a user chosen password. The key is computed by hashing the user’s password with SHA-256 to add more entropy and to make it harder to brute force the key. (Exhaustive search to find the key from the hash: 2^{256}. NIST defines infeasible to be 2^{112} [1]).

Approach	**Transmission speed**	**Network stealthiness**	**Physical stealthiness**
Malformed bluetooth packets	**High:** 1600bit/s (considering we send 10 packets/second	**Suspicious:** packets are malformed.	**Invisible:** The bulb won’t physically react to a malformed packet.
Color steganography	**Good:** 120bit/s (considering we send 10packets/second). It can be adjusted by tweaking the number of bits used to code the payload.	**Stealthy:** Use legitimate bluetooth commands to exfiltrate data.	**Stealthy:** Colors variations are invisible for human eye if no more than the 4 less significant bits per color channel are used to encode data.

The steganographic approach gives the best compromises between covertness and transmission speeds and could be exploited to leak sensitive data from an air gapped network.

V- <u>One web browser to rule them all</u>

Different software solutions can be considered to exploit the steganographic approach: We first have developed a python client. Despite being fully functional, the script did not conform to the initial considerations as it required third party libraries to connect with the bulb and admin rights to be executed. Moreover, according to those assumptions, we also cannot assume that the python interpreter will be installed on on-site workstations. Thus, a python client is not usable in this scenario and the restrictive environment makes the use of most classical approaches impractical.

So how do you send custom crafted commands to a smart bulb when you:

- Have no internet access (i.e. no third-party libraries / software installation)
- Have only the pre-installed software at your disposition (i.e. no IDE, dev environment, etc.…)
- Have no admin rights on the computer (i.e. almost anything you can think of won't work...)
- … ?

… By using the web browser !

Indeed, thanks to the new native web Bluetooth API [2], most of recent web browsers are capable of interacting with Bluetooth devices with a few lines of JavaScript. Moreover, web browsers are present by default on every computer and are used even in air gapped networks to access intranet resources. They are also available for any users without any permission, making them a perfect attack vector.

To take advantage of this, we have developed a JavaScript web app to exfiltrate sensitive data through the bulb. Our web app has been specially designed to work on highly secured networks and do not require any internet connection, any admin right to be executed, nor any third-party software installation. Indeed, unlike the python client that require python to be installed as well as an external Bluetooth library to control the bulb, the web app only needs to be opened with a web browser to work. (An online version of our web app can be found at http://www.corentin.thomasset.me/csaw).

To deploy the app inside the protected network, we have developed two specific variants of the online web app to bypass security measures (as it cannot be downloaded from the internet nor be copied from a storage device):

- **A minified version of the script that can be embedded in a QR-Code** and printed on a sheet of paper. This enable the attacker to bring the script in the restricted area and to recreate the web app using a QR-Code reader (IOS and Android devices comes with QR-Codes reader by default).

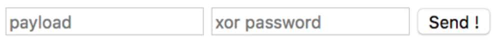
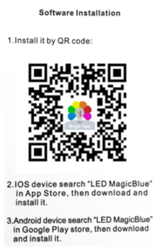


Figure 6: An exfiltration script embedded in a QR-Code and the resulting web app.

- **A fake magic blue control application** that can be downloaded and installed on a smartphone instead of installing the legitimate control app.


Software Installation

1.Install it by QR code:

2.IOS device search "LED MagicBlue" in App Store, then download and install it.

3.Android device search "LED MagicBlue" in Google Play store, then download and install it.

Figure 7: Fake app installation QR-Code

V- <u>Data exfiltration scenario</u>

Prerequisites:

- An attacker inside the air gapped network with some sensitive data to exfiltrate (eg: Cryptographic keys).
- A Magic Blue Bluetooth light bulb controllable by the attacker.
- An accomplice within the range of emission of the attacker with appropriate hardware to sniff the connection.

Attack flow:

1. The attacker set-up the web app inside the protected network to control the bulb using one of the variants described above. As our web app is cross platform, the only requirement is to find a device that can access the sensitive data and that supports BLE.

2. Once the web app is operational the attacker starts the data exfiltration process. Our app provides different options to improve the covertness of the attack and to make the data leak source identification harder: In addition to the steganographic method, the payload can be encrypted using a Xor cipher and the attacker can delay commands sending to evade detection.

3. The attacker's accomplices within the range of emission sniff the connection between the smart bulb and the attacker using a Bluetooth sniffer. By analyzing the Bluetooth traces, they recover the payload hidden in color change commands. As sniffers are passive devices, they cannot be detected.

4. To clear his tracks, the attacker deletes the html file packaging the web app and clear its entry in the browser history. As no firmware / hardware modifications have been made to the bulb and as the fingerprints on the attacker device are very limited aftermath forensics will likely be unable to identify the leakage source and the leaked data.

V- Proofs of concept - step by step executions

A - Data exfiltration using the web app embedded in a QR-Code

1- Create a new html file poc.html
2- Edit poc.html with your favorite IDE. Notepad will also do the job.
3- Insert the following code:

```
<html>
    <input type="text" placeholder="payload" id="a"/>
    <input type="text" placeholder="xor password" id="p"/>
    <button onclick="send()">Send !</button>
    <script>
     // Insert the QR-Code text here
    </script>
</html>
```

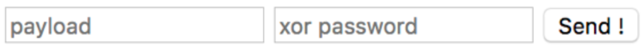
4- Scan the QR-Code

5- Paste the text from the QR-Code between the script tags.

6- Save the file

7- Open it in Chrome

8- Type or paste the payload and the Xor password in the inputs



9- Press send

10- Pick the light bulb in the Bluetooth pairing pop-up

11- Done ! The payload is sent to the bulb.

B - Data exfiltration using the fake magic blue application

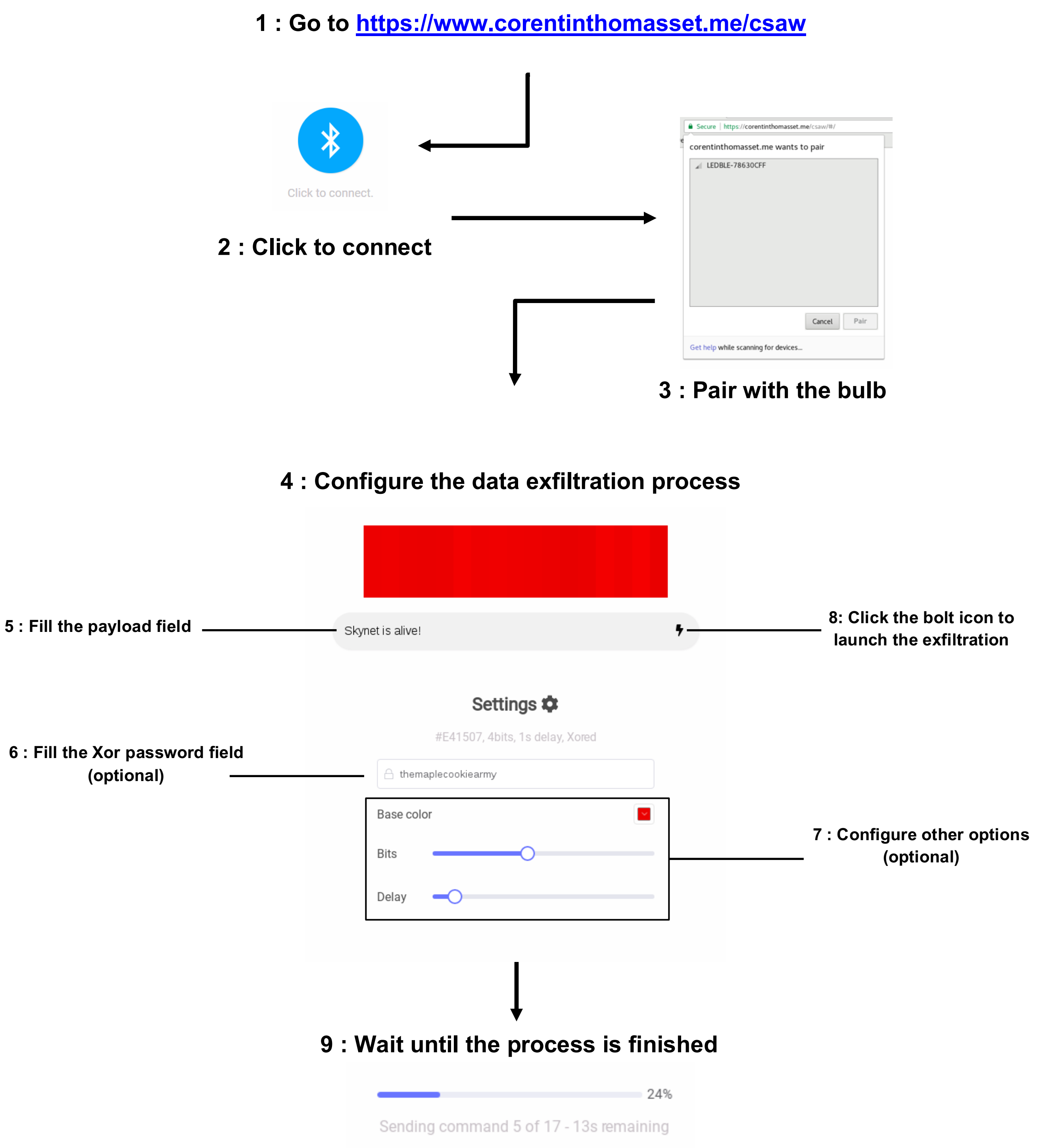
The fake magic blue application is a progressive web app version of our web app that can be installed for offline use on mobile devices.

1- Install the progressive web application on a smartphone with the following QR-Code:

2- Follow the instruction on how to exfiltrate data using the online application.

C - Data exfiltration using the online web application

1- An online version of the web app is available at https://corentinthomasset.me/csaw.
2- Click on the Bluetooth icon to open the paring pop-up.
3- Select the Magic Blue Bulb and click on “pair”.
4- The bulb is now paired, and you are redirected to the control page.
5- Type the payload in the payload field.
6- Type the Xor password in the Xor key input field (optional).
7- You can adjust some parameters to be stealthier as the base color from which we encode the data, the number of least significant bit used to encode the payload and the delay between commands.
8- Click the bolt icon in the payload field to start exfiltrating data !
9- The progress bar indicates the status of the process and keep you informed of the remaining time.


1 : Go to https://www.corentinthomasset.me/csaw
Click to connect.
2 : Click to connect
Secure | https://corentinthomasset.me/csaw/#/
Cancel
Pair
Get help while scanning for devices...
3 : Pair with the bulb
4 : Configure the data exfiltration process
5 : Fill the payload field
8: Click the bolt icon to launch the exfiltration
Settings
6 : Fill the Xor password field (optional)
Base color
Bits
Delay
7 : Configure other options (optional)
9 : Wait until the process is finished
24%

corentinthomasset.me wants to pair

LEDBLE-78630CFF

Skynet is alive!

#E41507, 4bits, 1s delay, Xored

themaplecookiearmy

Sending command 5 of 17 - 13s remaining

D - Data exfiltration using the python script

Prerequisites:
You must use linux (debian or fedora) and have python 3+ installed as well as a proper Bluetooth 4.0 interface.

If you are using debian:

- libglib2.0-dev is necessary (sudo apt-get install libglib2.0-dev)

if you are using fedora:

- glib2-devel is necessary (sudo dnf install glib2-devel)

To exflitrate the data, use the “code_and_send.py” script:

sudo ./code_and_send.py --message “MESSAGE” --bulb_macaddr MACADDR --key KEY --nbbit NBBIT --base_color “R G B”

Where message and bulb_macaddr are required argument. Other are optionals.

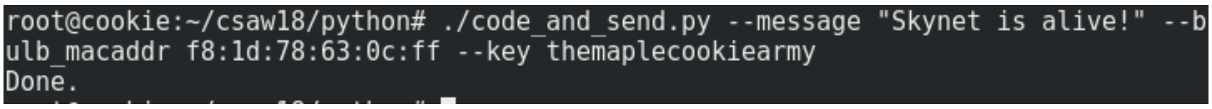
- “MESSAGE”: should be replaced by the message to transmit
- MACADDR: bulb mac address (ex: f8:1d:78:63:0c:ff).
- KEY: the key to use for the Xor encryption, if not specified, the key will be: “themaplecookiearmy”.
- NBBIT: the number of least significant bits to use to hide the data. If not specified, 4 bits will be used.
- “R G B”: the base color to use for the steganography. If not specified, white is used (“255 255 255”).

Once the payload has been sent, the script should print “Done.”.

Example to exfiltrate ‘Skynet is alive!’:

To exfiltrate “Skynet is alive!” through a bulb (f8:1d:78:63:0c:ff) with “themaplecookiearmy” for Xor password, use the following command:

sudo ./code_and_send.py --message “Skynet is alive!” --bulb_macaddr f8:1d:78:63:0c:ff --key themaplecookieamy

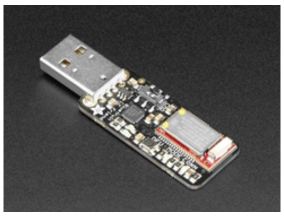


```
root@cookie:~/csaw18/python# ./code_and_send.py --message "Skynet is alive!" --b
ulb_macaddr f8:1d:78:63:0c:ff --key themaplecookiearmy
Done.
```

E - Capturing bluetooth packets using adafruit Bluefruit LE Sniffer

Prerequisites:

- An Adafruit Bluefruit LE Sniffer:
- Wireshark software
- Administrator rights



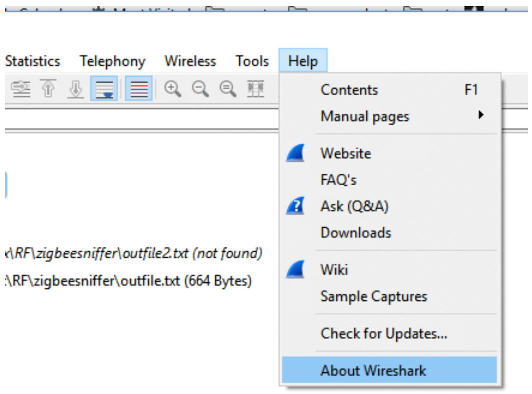
Capturing packets:

1- Install the following drivers:

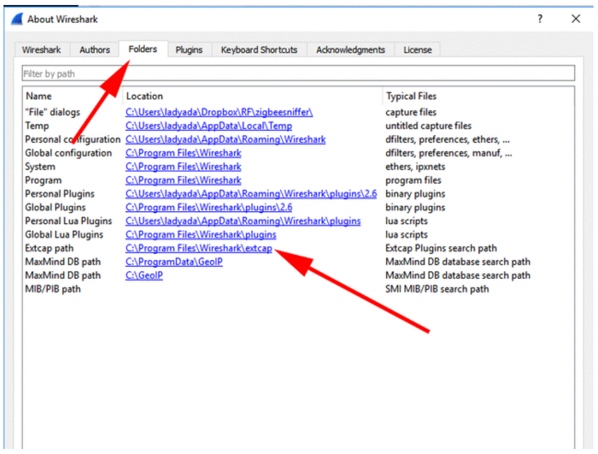
- https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers
- https://www.ftdichip.com/Drivers/VCP.htm
-

2- Install the Wireshark plugin:
https://www.nordicsemi.com/eng/nordic/download_resource/65225/1/81282695/136163

3- Open wireshark, in the **Help** menu select **About wireshark**

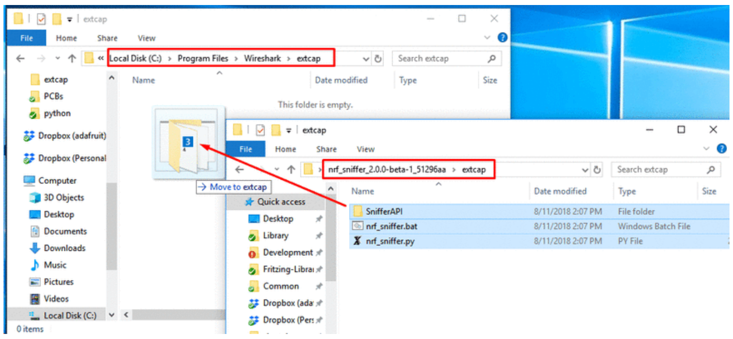


4- In the **Folders** tab, find the extcap path



Name	Location	Typical Files
"File" dialogs	C:\Users\ladyada\Dropbox\RF\zigbeesniffer\	capture files
Temp	C:\Users\ladyada\AppData\Local\Temp	untitled capture files
Personal configuration	C:\Users\ladyada\AppData\Roaming\Wireshark	dfilters, preferences, ethers, ...
Global configuration	C:\Program Files\Wireshark	dfilters, preferences, manuf, ...
System	C:\Program Files\Wireshark	ethers, ipxnets
Program	C:\Program Files\Wireshark	program files
Personal Plugins	C:\Users\ladyada\AppData\Roaming\Wireshark\plugins\2.6	binary plugins
Global Plugins	C:\Program Files\Wireshark\plugins\2.6	binary plugins
Personal Lua Plugins	C:\Users\ladyada\AppData\Roaming\Wireshark\plugins	lua scripts
Global Lua Plugins	C:\Program Files\Wireshark\plugins	lua scripts
Extcap path	C:\Program Files\Wireshark\extcap	Extcap Plugins search path
MaxMind DB path	C:\ProgramData\GeoIP	MaxMind DB database search path
MaxMind DB path	C:\GeoIP	MaxMind DB database search path
MIB/PIB path		SMI MIB/PIB search path

5- Open that directory up, then copy over the files within the nrf_sniffer.zip extcap folder into the extcap folder

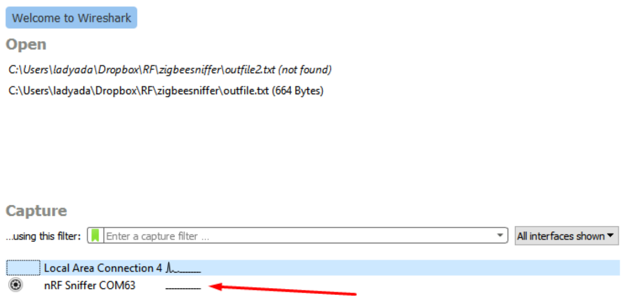

extcap
File
Home
Share
View
Local Disk (C:) › Program Files › Wireshark › extcap
Search extcap
This folder is empty.
Move to extcap
Quick access
nrf_sniffer_2.0.0-beta-1_51296aa › extcap
0 items

Name	Date modified	Type	Size
SnifferAPI	8/11/2018 2:07 PM	File folder	
nrf_sniffer.bat	8/11/2018 2:07 PM	Windows Batch File	
nrf_sniffer.py	8/11/2018 2:07 PM	PY File	

6- Nordic’s sniffer is using python2 code, if you have Python 3 as default, please install python2 and set it up as your default python.
7- Install dependencies:

- pip2 install pyserial

8- Start Wireshark, the nRF Sniffer capture device should appear



Welcome to Wireshark

Open

C:\Users\ladyada\Dropbox\RF\zigbeesniffer\outfile2.txt (not found)
C:\Users\ladyada\Dropbox\RF\zigbeesniffer\outfile.txt (664 Bytes)

Capture

...using this filter: Enter a capture filter ... All interfaces shown

Local Area Connection 4
nRF Sniffer COM63

9- Double click on that line to start the capture !

Please refer to the official installation guide if any problem occurs:
https://learn.adafruit.com/introducing-the-adafruit-bluefruit-le-sniffer/introduction

F - Capturing Bluetooth packets using Wireshark

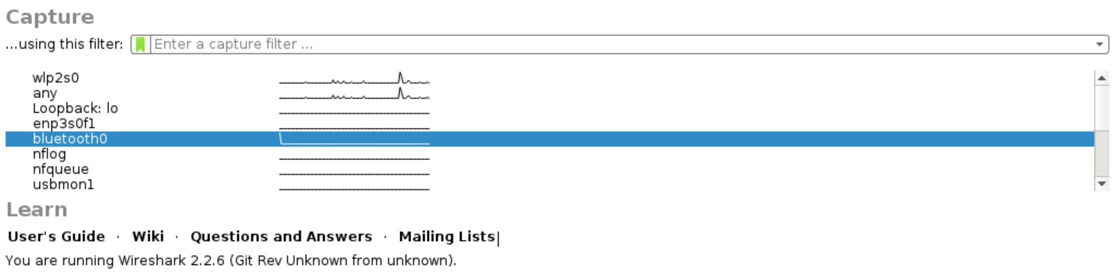
Prerequisite:

- Wireshark installed: https://en.wikiversity.org/wiki/Wireshark/Install.
- Administrator rights.

To capture Bluetooth traffic:

1- Launch Wireshark

2- Select the Bluetooth interface by double clicking on it (Here, bluetooth0):



Capture

...using this filter: Enter a capture filter ...

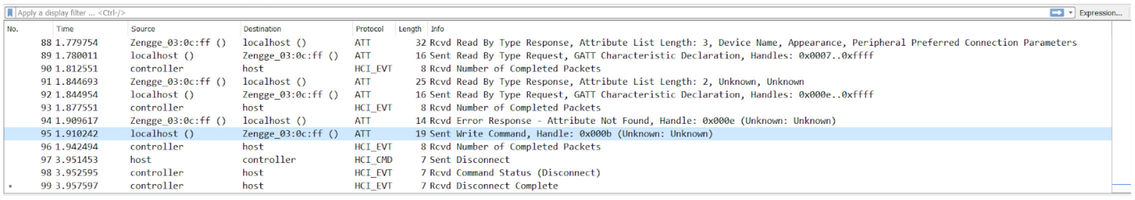
```
wlp2s0
any
Loopback: lo
enp3s0f1
bluetooth0
nflog
nfqueue
usbmon1
```

Learn

User's Guide · Wiki · Questions and Answers · Mailing Lists

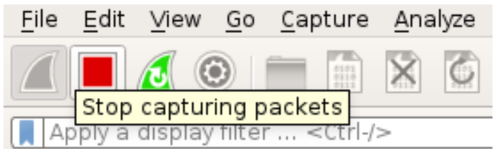
You are running Wireshark 2.2.6 (Git Rev Unknown from unknown).

3- The capture starts automatically, and captured packets are displayed in the main window:



```
No.   Time       Source              Destination         Protocol  Length Info
   88 1.779754   Zengge_03:0c:ff ()  localhost ()        ATT       32 Rcvd Read By Type Response, Attribute List Length: 3, Device Name, Appearance, Peripheral Preferred Connection Parameters
   89 1.780011   localhost ()        Zengge_03:0c:ff ()  ATT       16 Sent Read By Type Request, GATT Characteristic Declaration, Handles: 0x0007..0xffff
   90 1.812551   controller          host                HCI_EVT    8 Rcvd Number of Completed Packets
   91 1.844693   Zengge_03:0c:ff ()  localhost ()        ATT       25 Rcvd Read By Type Response, Attribute List Length: 2, Unknown, Unknown
   92 1.844954   localhost ()        Zengge_03:0c:ff ()  ATT       16 Sent Read By Type Request, GATT Characteristic Declaration, Handles: 0x000e..0xffff
   93 1.877551   controller          host                HCI_EVT    8 Rcvd Number of Completed Packets
   94 1.909617   Zengge_03:0c:ff ()  localhost ()        ATT       14 Rcvd Error Response - Attribute Not Found, Handle: 0x000e (Unknown: Unknown)
   95 1.910242   localhost ()        Zengge_03:0c:ff ()  ATT       19 Sent Write Command, Handle: 0x000b (Unknown: Unknown)
   96 1.942494   controller          host                HCI_EVT    8 Rcvd Number of Completed Packets
   97 3.951453   host                controller          HCI_CMD    7 Sent Disconnect
   98 3.952595   controller          host                HCI_EVT    7 Rcvd Command Status (Disconnect)
   99 3.957597   controller          host                HCI_EVT    7 Rcvd Disconnect Complete
```

4- Click on the red square to stop the capture.



4- Save the capture with the “File > Save”.

G - Decoding the payload

Prerequisites:
You must use linux (debian or fedora) and have python 3+ installed as well as a proper Bluetooth 4.0 interface.

If you are using debian:

- libglib2.0-dev is necessary (sudo apt-get install libglib2.0-dev)

if you are using fedora:

- glib2-devel is necessary (sudo dnf install glib2-devel)

To decode the payload, use the “decode.py” script:

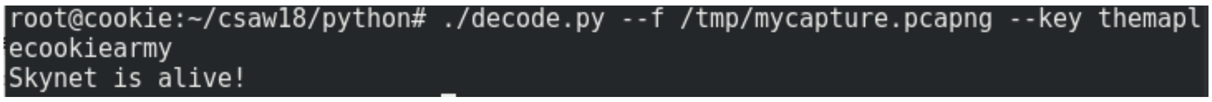
```
./decode.py --f FILE --nbbit NBBIT --key KEY --qrcode
```

Where file is a required argument, other are optionals.

- FILE is the path to the Bluetooth traffic capture to decode
- NBBIT is the number of least significant bit used to hide the data. If not specified, 4 is used by default.
- KEY: the key used to decode the Xor encryption.
- --qrcode is an optional flag, use it only if data was exfiltrated via the qrcode method.

Example with “Skynet is alive!”:
To decode a Bluetooth capture located in /tmp/mycapture.pcapng to recover the message, do:

```
sudo ./decode.py --f /tmp/mycapture.pcap --key themaplecookieamy
```



```
root@cookie:~/csaw18/python# ./decode.py --f /tmp/mycapture.pcapng --key themapl
ecookiearmy
Skynet is alive!
```

H- Troubleshooting:

Web apps have been tested to work on chrome and chromium web browsers.
As the bluetooth web API is still an experimental feature, you may need to activate it through the flag: *chrome://flags/#enable-experimental-web-platform-features.*

VI- Final thoughts

As shown previously our approach using a web browser to control the bulb to leak sensitive data could be used in a real-world attack against an air gapped network. Indeed, our approach has a lot of advantages making it a solution of choice for very restrictive environments:

- It doesn't require any internet access.
- The exfiltration process doesn't produce any traffic on the physical network and the Bluetooth traffic resulting is not suspicious as it is only composed of legitimate commands.
- It doesn't require any hardware / firmware modification on the IoT device making it easier to set-up and harder to investigate afterward.
- The usage of a web app executed in a web browser doesn't require any third-party library / software installation and can be ran without any privilege.
- The html file embedding the application is also not as suspicious as executable files or scripting languages files.
- The web app is cross platform and can be executed on any device with a Bluetooth adapter and a web browser (Workstation, smartphones, tablets). Most of those devices comes with a web browser installed by default.
- The web app also provides a simple graphical interface enabling any employee to run the attack without any advanced knowledge in computer sciences.
- As its source code is really brief, the web app can be easily deployed using a paper printed QR-Code to bypass TSA like security screenings.
- The transmission speed is fast enough to leak cryptographic key in a reasonable amount of time.
- The range of the attack is only limited by the range of emission of the attacker device. This can be increased by using a device with a more powerful Bluetooth adapter and some specialized hardware on the receiver side (bluesniper [3]).

VII - Other work

A) Optical side channel attacks

During the development of our solution, we've explored optical side channel attacks. They are very popular attacks on smart bulbs and are highly covered by research papers. We first tried to exploit infra-red emissions of the bulb that can't be seen by the human eye but could be detected by a camera. Some bulbs have dedicated infrared LEDs and we were curious to see if the Magic Blue Bluetooth Bulb would also emit in those frequencies. We tried to make an infrared filter with old floppy disk (that's a the poor way of doing it..). Unfortunately, this bulb doesn't have any infrared LEDs and its emission in IR frequencies was very limited making it hard to exploit.

Our second idea was to use lights variations such as brightness or color change to transmit data. Those variations are detected and interpreted by the receiver outside the network to reconstruct data. The most common way to make light variation invisible to the human eye is to use high frequency flickering (Flickers over 60Hz cannot be detected by a human). After identifying the commands to control the bulb, we have created a script to send as many commands as possible in a short time lapse. Unfortunately, in our case, the bulb was not able to handle so many commands making it impossible to exploit the high frequency flickering.

B) Hardware attacks

We've tried to use the bulb as a storage device. As bulbs are easily accessible by employees, we thought of using it to save data and discreetly steal one by replacing it with another.
We have also noticed that the light saves its state. As a program can be composed with up to 16 different colors and each color is coded on 3 bytes, it allows us to exfiltrate only 48 bytes data per bulb (as a bulb can only store one program). This would be complicated to use when trying to exfiltrate a huge file or cryptographic keys.
To increase the bandwidth, we have tried to abuse color inputs in order to write data to unexpected memory plages and extend the bulb capacity. We did it by sending huge and unexpected Bluetooth packets to the light. Unfortunately, the light doesn't seem to interpret those packets as it didn't crash or react either.

References

[1] NIST Special Publication 800-133: Recommendation for Cryptographic Key Generation - Elaine Barker, Allen Roginsky https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-133.pdf
[2]Interact with Bluetooth devices on the web - Google Developers - François Beaufort https://developers.google.com/web/updates/2015/07/interact-with-ble-devices-on-the-web
[3] How to: Building a BlueSniper Rifle - Part 1https://www.smallnetbuilder.com/wireless/wireless-howto/24256-howtobluesniperpt1