GATTACKING
BLUETOOTH SMART DEVICES

Sławomir Jasek, SecuRing (slawomir.jasek@securing.pl)

TABLE OF CONTENTS

Abstract 2
1. Bluetooth Low Energy 2
  1.1. BLE devices 2
2. BLE COMMUNICATION 2
  2.1. Broadcast advertisement 2
  2.2. Listening for advertisements 3
  2.3. Connection to device 3
  2.4. GATT data structure: services, characteristics, descriptors 3
  2.5. Browsing device's services 4
  2.6. Reading, writing and notifications 4
3. BLE SECURITY 5
  3.1. BLE security - specification 5
    3.1.1. Encryption 5
    3.1.2. Random MAC address 5
    3.1.3. Whitelisting 5
  3.2. BLE security - practice 5
4. POSSIBLE ATTACKS 6
  4.1. Attacks on advertisements 6
    4.1.1. Example vulnerabilities 6
    4.1.2. Attack countermeasures 7
  4.2. Passive Interception 7
    4.2.1. Example vulnerabilities 7
  4.3. Active interception 8
    4.3.1. Example vulnerabilities 9
    4.3.2. Attack countermeasures 10
  4.4. Attacks on exposed services 10
    4.4.1. Example vulnerabilities 10
    4.4.2. Attack countermeasures 11
  4.5. Attacks on pairing 11
    4.5.1. "Just Works" 11
    4.5.2. PIN-protected pairing 11
    4.5.3. Attack countermeasures 11
  4.6. Whitelisting bypass 12
  4.7. Privacy considerations 12
5. ATTACK CONDITIONS, RISK CONSIDERATION 12
  5.1. Physical range 12
  5.2. Risk 12
6. THE NEW TOOL 12
  6.1. Architecture 12
  6.2. Implementation 12
  6.3. Necessary hardware 13
  6.4. Device communication analysis 13
    6.4.1. Mobile application analysis 13
    6.4.2. HCI dump 13
7. REFERENCES 13

ABSTRACT

This document outlines possible forms of a Bluetooth Low Energy attack. Special attention has been paid to the higher, GATT (Generic Attribute Profile) layer of the Bluetooth stack. The introduction consists of the fundamental attributes of BLE. This section is followed by a breakdown of possible risks, attack scenarios and suggested countermeasures. The attack scenarios are complemented by several real-life vulnerabilities, which were identified during the research phase in tested devices and accompanied mobile applications.

Ultimately, a new open source tool is introduced, which assists in the security assessment of BLE devices.

1. BLUETOOTH LOW ENERGY

As its name implies, Bluetooth Low Energy (also known as Bluetooth Smart or Bluetooth 4) technology, was designed from its inception to be power-efficient. According to some manufacturers' claims, the BT4 chip can operate on a single coin battery for "months to years" (depending on usage and power configuration levels), although our testing could not replicate these results. Besides having "Bluetooth" in the name, the BLE protocol does not share much more with previous Bluetooth versions (also called BR, EDR, 1.2, 2, 3...). This version has a new RF stack (although it still operates on 2.4 GHz ISM band), and utilizes other usage scenarios. Focus has been put on simplicity rather than throughput, thus making the chip not only less energy hungry, but also significantly smaller and cheaper. And this key characteristic turned out to be the catalyst for the explosion of a wide assortment of new "IoT" devices and applications on the market.

1.1. BLE devices

The availability, low cost and ease of implementation has rendered the technology extremely popular among startups, which develop hundreds of varying "smart" BLE-enabled products. Of course, crowdfunding projects are just a slice of actual implementation, as BLE is also making its way into medical, industrial and government equipment. It is forecasted that more and more BLE devices will surround our lives in the form of wearables, sensors, lightbulbs, socks, cups, medical devices, and other smart-products. Many of these connected devices are not associated with any significant risk, but some may possess serious security implications (i.e. door locks, alarms, security sensors, biometric authentication, banking tokens, keypasses etc.). Also many devices expose users to potential privacy vulnerabilities.

2. BLE COMMUNICATION

Bluetooth Low Energy communication between device and mobile application follows usually a scheme:

1. Device (peripheral) broadcasts an advertisement.
2. Central device (mobile phone) scans for advertisements.
3. Once the specific advertisement packet is received, the central device stops scanning, and initiates a connection to the broadcasting peripheral.
4. Central device browses the peripheral device for available services.
5. Central device exchanges information with peripheral device using characteristic read/write/notify requests and responses.

Depending on the usage scenario, the mobile application may handle only advertisements (2), without initiating direct connection to the peripheral device.

A detailed description of each step follows.

2.1. Broadcast advertisement

The broadcasting device advertises packets with a specified interval and TX power level. On the RF layer, the advertisements are broadcasted using 3 dedicated channels (out of 40 2 MHz-wide channels the 2.4 GHz ISM band is split into by BLE), with frequencies optimized to avoid Wi-Fi interference. The device may choose which channels to use for advertising: selected either 1, 2 or most commonly all 3 of them.

The packets are very limited in size (31 bytes), and formatted according to specification defined by Bluetooth SIG [1].

In consecutive fields, the device may broadcast, i.a. its "services", "name", or "manufacturer data" (field type 0xFF). The manufacturer data can be formatted according to other widely recognized formats (not defined by Bluetooth SIG), e.g. Apple iBeacon or Google Eddystone. Vendors can also implement their own, proprietary data format.

On a lower layer, advertisement data can be split into 2 packets - one broadcasted by the device independently, and the second a “scan response” - sent back to a specific scanning device in response to a scan inquiry request.

The broadcast packet is by design visible to all listening devices in range (with exception of not widely adopted “directed advertising” mode). The broadcast is used mostly to “advertise” device presence to mobile applications, as well as transmit non-private data, device status or sensor indication.

2.2. Listening for advertisements

The “central” device (usually a smartphone), switches into scanning advertisement mode. In this mode, it receives all the advertisements of nearby devices. Next, the mobile application matches the received advertisements against a specific one, related to given device.

As the scanning requires a significant amount of power, in order to conserve the battery, the scan process is usually stopped immediately after receiving the first matching advertisement.

Next, the mobile application interprets the received data, and performs the suitable actions. In several scenarios (e.g. beacons, some sensors, getting a device’s status), the mobile application does not need to initiate further connection to device.

2.3. Connection to device

If the usage scenario requires exchanging more data with the device, the directed connection is initiated. The connection attempt is performed usually to the MAC address of a device with a matching advertisement. Depending on the mobile application, the MAC address may however be compared with the specific previously stored MAC (e.g. matching a given lightbulb), or a defined vendor class. As an example, follow the decompiled Android source code filtering MAC addresses for a specific vendor:

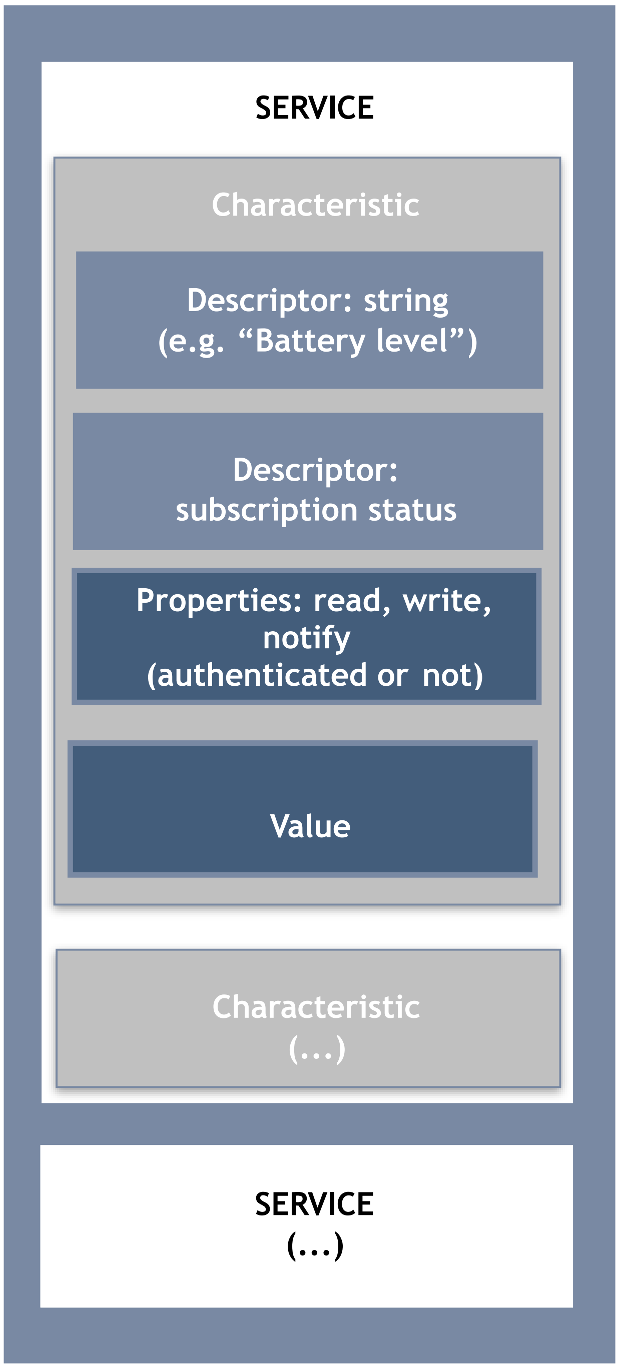
```
private static boolean
isBlueRadiosModuleAddress(String
paramString)
  {
    int i = paramString.substring(0,
8).compareTo("EC:FE:7E");
    boolean bool = false;
    if (i == 0) {
      bool = true;
    }
    return bool;
  }
```

Most devices allow for only one active connection at a time.

2.4. GATT data structure: services, characteristics, descriptors

Devices exchange data using General Attribute Profile (GATT) [2] characteristics, descriptors and services. The figure below depicts their relationship:

GATT data structure: services, characteristics, descriptors


SERVICE
Characteristic
Descriptor: string (e.g. “Battery level”)
Descriptor: subscription status
Properties: read, write, notify (authenticated or not)
Value
Characteristic (…)
SERVICE (…)

A characteristic contains a single value ("attribute"), which can be read, written to or subscribed for notifications (details in chapter 2.6).

Each service and characteristic is identified by an associated UUID (Universally Unique Identifier). Typical services (e.g. battery level, device information) use short UUID values defined in the Bluetooth specification [3].

To create their own proprietary services and characteristics vendors have to define their own long UUID values.

Example: the proprietary UUID service and characteristic values of Apple Watch, as explored by GATTacker tool:

```
    "uuid":
"d0611e78bbb44591a5f8487910ae4366",
        "name": null,
        "type": null,
        "startHandle": 10,
        "endHandle": 14,
        "characteristics": [
            {
                "uuid":
"8667556c9a374c9184ed54ee27d90049",
                "name": null,
                "properties": [
                    "write",
                    "notify",

"extendedProperties"
                ],
                "value": "",
                "descriptors": [
                    {
                        "handle": 13,
                        "uuid":  "2900",
                        "value": ""
                    },
                    {
                        "handle": 14,
                        "uuid":  "2902",
                        "value": ""
                    }
                ],
                "startHandle": 11,
                "valueHandle": 12
            }
        ]
```

A characteristic can have associated descriptors. Possible descriptor types are defined in the according specification [4].

The two most commonly used descriptors are: 0x2901 (human readable user description), and 0x2902 - "client characteristic configuration", which describes the current subscription status.

2.5. Browsing device's services

After initiating a connection, the central device scans the peripheral for all available services, characteristics and descriptors.

As the services scanning process takes several requests and responses, mobile operating systems store cached values for specific devices, in order to optimize the process. For example, Android operating system stores GATT cache in /data/misc/bluedroid: bt_config.xml and gatt_cache_<MAC_ADDR> files.

2.6. Reading, writing and notifications

Reading and writing to characteristics is performed according to the Generic Attribute Profile (GATT), which defines a structured list of the services, characteristics and attributes of a given application.

As mentioned earlier, each characteristic has associated properties defining its possible actions: read, write, notify. The properties can be used separately or in unison (e.g. read+write, write+notify, read+write+notify).

Each action may also require "authentication", meaning encryption of the connection (and usually those devices are paired). In such a case, the initial read or write request is followed by "insufficient authorization" response from the device. Once the devices establish an encrypted connection the consecutive read or write requests to such characteristic proceed normally.

Read and write requests transmit a single value. For getting more data or receiving periodic updates from a device, notifications are used. The central device subscribes for a specific characteristic, and the peripheral device sends data asynchronously.

Technically, subscription is performed as a write request to a dedicated descriptor (0x2902). Reading this descriptor value returns the current subscription status.

A write request can be with or without a response, and a notification can be unconfirmed or confirmed by the recipient (also called "indication").

The low-level communication is actually performed using integer handle numbers, associated with specific characteristics.

3. BLE SECURITY

3.1. BLE security - specification

According to specification [5], Bluetooth Low Energy “provides several features to cover the encryption, trust, data integrity and privacy of the user’s data”.

3.1.1. Encryption

In order to encrypt transmission, BLE devices undergo a pairing procedure. During this process they set up a Long Term Key, used then to secure consecutive connections. The available options include:

- “Just Works”
- Passkey Entry
- Out Of Band

Version 4.2 of the Bluetooth specification introduces elliptic curves as an addition. At the time of writing this whitepaper, devices supporting this version of protocol are not yet widespread.

The pairing method should be selected depending on the device input/output capabilities (display, yes/no button, keyboard). For example, devices without a display obviously cannot use Passkey Entry. The first two options are most common, Out Of Band is not widely adopted.

Citing the specification: “Just Works and Passkey Entry do not provide any passive eavesdropping protection”. Sniffing the pairing process allows for deriving the Long Term Keys from the PIN values, and consequently decrypting the transmission. In the case of “Just Works” the static PIN value used is: 000000. The Passkey Entry PIN entry value can be brute-force cracked using the Crackle tool [6].

Although the most commonly used pairing options are susceptible to passive interception, the idea is that it is supposed to be performed only once and in a secure environment. And after the initial bond is created, there transmission is properly secured using “Long Term Keys”.

3.1.2. Random MAC address

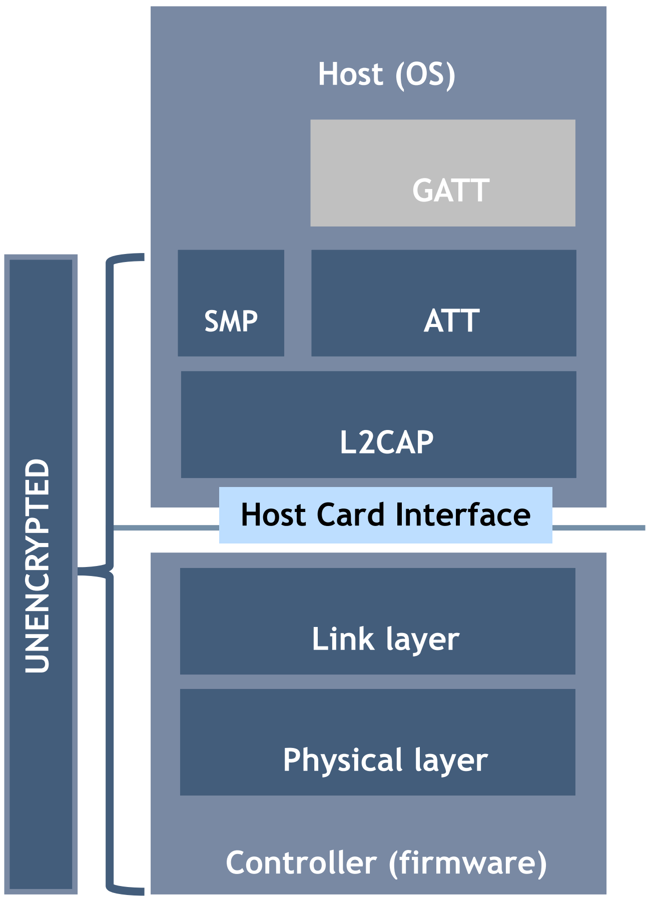
In order to prevent tracking, the specification allows the change of the MAC address of the device on a frequent basis. Only a paired device is able to resolve the current MAC.

3.1.3. Whitelisting

It is possible to create a whitelist of accepted devices' MAC addresses.

3.2. BLE security - practice

A significant amount of devices do not implement the abovementioned security features. For many device's usage scenarios (e.g. cash registers, devices with remote sharing feature, managing a “fleet” of beacons) it is not possible to carry out the pairing procedure in a secure environment. Some vendors do not associate any significant risk with the possibility of intercepting the transmission, and so they accept it. Others struggle to comply with various requirements: usability, multiple users or devices, cloud backup etc. The Bluetooth security features are handled by an operating system, and the mobile application does not have full control over this process. It is not easy to share access or to transfer it to another device. This is why these developers have decided to create their own security mechanisms on top of the unencrypted Bluetooth LE link, using the GATT read/write/notify requests. The most common features include secure authentication (mostly following challenge-response scheme), and data encryption. Usually, only hardware supported algorithm - AES - is used, in combination with their own proprietary protocols.



With the exception of smartphones and smartwatches, MAC randomization is currently not very widely adopted. And even if the device declares “random” MAC type, it often does not switch it on a regular basis. Randomization can also cause problems with the whitelisting implementation, which is also uncommon.

4. POSSIBLE ATTACKS

4.1. Attacks on advertisements

A mobile application that interprets advertisements broadcasted by a device can be attacked by advertisement spoofing.

Most battery-powered devices optimize advertising intervals in order to minimize their power consumption. In attack scenarios that include “jamming” the original device, an attacker may abuse quality by broadcasting advertisements with the minimum possible intervals - much more frequently than the original device. As described in LINK 2.2, the mobile application will interpret the first received advertisement - and in this case it will most probably be the spoofed one.

Additionally, as most devices do not broadcast advertisements during active connection, by maintaining the connection with the original device, it is possible to prevent it from broadcasting.

Advertisement spoofing is made possible using the newly introduced tool’s features to record all broadcasted packets, and then to advertise them with configurable (by default minimal) interval. Whenever possible, it simultaneously maintains a connection to original device.

The simplest possible attack is Denial of Service. In order to successfully execute it, all that is necessary is to advertise the “cloned” device, without even setting up corresponding services. The victim's mobile application will try to connect to it, and not being able to access the needed device functions, and will start scanning for advertisements once again. The attack is even more efficient with the cloned services set, but without forwarding the requests to the original device. In such a case, the victim's mobile application stays connected longer to the cloned device before it tries to re-connect.

4.1.1. Example vulnerabilities

Home automation Denial of Service

Example: home automation mobile application had its status to associated connected devices (lightbulbs, smart plugs etc.) interrupted via advertisement packets. By spoofing the device state in the advertisement packets - e.g. advertising its status as “off” while in fact device was “on” - it was possible to block the application’s functionality. In effect, the attacked user was unable to control the device using the mobile application.

For this application, the advertisement signal had to be broadcasted from a matching MAC address of the attacked device, as the mobile application stored specific devices by their MACs. Therefore, the attack involved cloning the MAC address of the attacked device.

Anti-theft proximity

Here is an example of the anti-theft functionality of a mobile application serving a luggage locking device, which depended on the availability of specific advertisement packets broadcasted by the device.

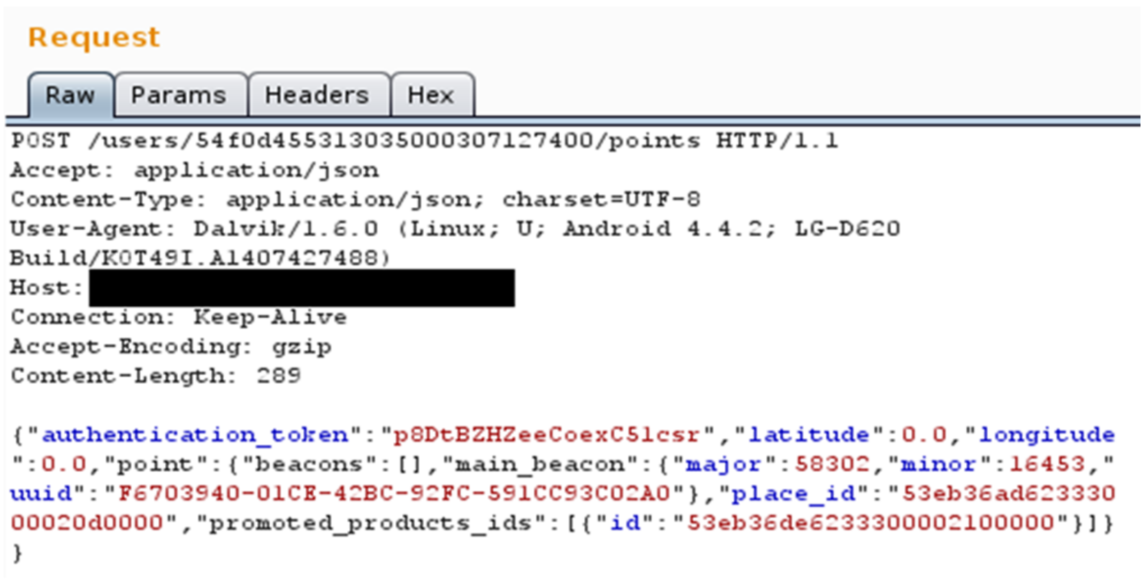
The attack scenario relied on simply spoofing advertisement packets broadcasted by the device, using its MAC address. The anti-theft mobile application did not notice that the advertisements were spoofed, and as a result it was possible to “compromise” the luggage which was thought to be protected.

Beacon abuse

Here is an example of a mobile application that awarded users with loyalty points while visiting specific places. After collecting a certain amount of points, it was possible to exchange them for free services. The visits were confirmed automatically by the mobile application after receiving specific iBeacon data, broadcasted by the beacon device which was located onsite.

By simply spoofing the iBeacon data, it was possible to obtain points without going to actual location.

In this case, the attack could be simplified to repeating the HTTP request sent to the server-side API by mobile application during the process. The iBeacon UUID, Major and Minor specific numbers were sent among request parameters:



```
POST /users/54f0d455313035000307127400/points HTTP/1.1
Accept: application/json
Content-Type: application/json; charset=UTF-8
User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.4.2; LG-D620
Build/K0T49I.A1407427488)
Host: [redacted]
Connection: Keep-Alive
Accept-Encoding: gzip
Content-Length: 289

{"authentication_token":"p8DtBZHZeeCoexC5lcsr","latitude":0.0,"longitude
":0.0,"point":{"beacons":[],"main_beacon":{"major":58302,"minor":16453,"
uuid":"F6703940-01CE-42BC-92FC-591CC93C02A0"},"place_id":"53eb36ad623330
00020d0000","promoted_products_ids":[{"id":"53eb36de6233300002100000"}]}
}
```

The GPS position sent among parameters of this request could also be easily spoofed, and therefore should not be used as a reliable form of protection.

The availability of beacon-mapping sites (for example http://wikibeacon.org/map), which allow for locating beacon signals in a specific location, may make the attack even easier to invoke remotely.

4.1.2. Attack countermeasures

In order to prevent advertisement abuse, beacon vendors introduced "shuffling" (also called "encrypting") and signing options for the broadcasted values. The advertised packets change their values with predefined frequency, and the value is possible to "decode" only using the vendor's mobile application. However, such mechanisms have to overcome several limitations - on both the hardware and software side, as well as compromise for offline usage requirements. As vendors guard the "shuffling" algorithm's technical details in the way of top-secret intellectual property, it may raise concerns whether the mechanism was properly reviewed by a professional cryptographer.

Depending on the level of risk, the ideal solution would be to not rely on received advertisement packets for critical functionality.

4.2. Passive Interception

Unencrypted transmission can be intercepted by a passive eavesdropper. Bluetooth interception does not require sophisticated nor expensive hardware any more. There are several affordable hardware options which help to accomplish this task, including open-sourced Ubertooth by Great Scott Gadgets [7].

For the purpose of this research, a simple USB dongle based on the nRF51822 Nordic BLE module was used. At the time of this writing, it was available for $29.95 on the producer's website [8].

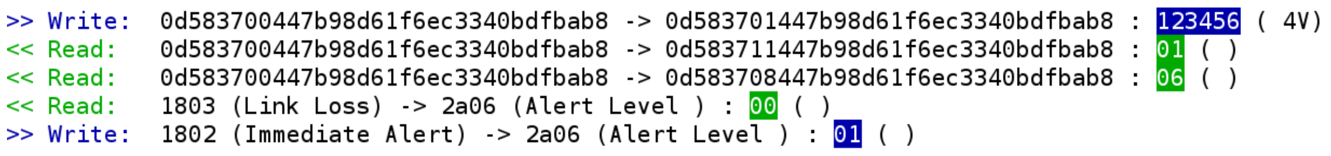
The device comes with software which feeds the sniffed packets to the Wireshark network analyzer.

Interception of transmitted data is also possible by active attack, using the newly introduced tool.

4.2.1. Example vulnerabilities

Smart finder

An example "smart finder" device implemented authentication in a form of a static 6-digit password sent from the mobile application to the device in cleartext form characteristic write. A screendump below presents the password ('123456') intercepted using the GATTacker tool:



```
>> Write:  0d583700447b98d61f6ec3340bdfbab8 -> 0d583701447b98d61f6ec3340bdfbab8 : 123456 ( 4V)
<< Read:   0d583700447b98d61f6ec3340bdfbab8 -> 0d583711447b98d61f6ec3340bdfbab8 : 01 ( )
<< Read:   0d583700447b98d61f6ec3340bdfbab8 -> 0d583708447b98d61f6ec3340bdfbab8 : 06 ( )
<< Read:   1803 (Link Loss) -> 2a06 (Alert Level ) : 00 ( )
>> Write:  1802 (Immediate Alert) -> 2a06 (Alert Level ) : 01 ( )
```

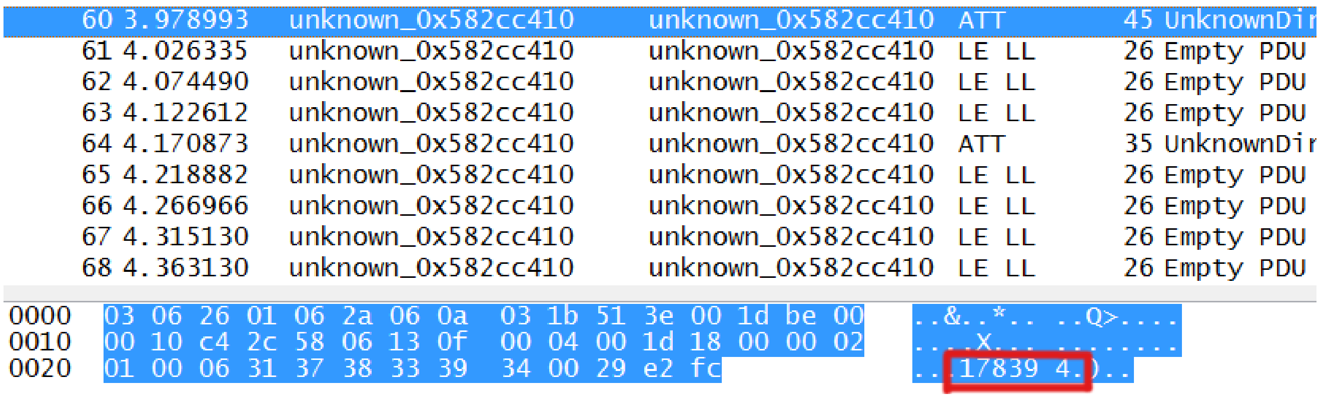
Beacon management

Beacon devices are usually managed via static password. Each device has its own individual password configured, which is delivered to mobile application via the server-side API. Next, in most cases the password is sent in a clear-text form to device.

OTP authentication token

A “One Time Password” demo token device was examined, which offered mobile application authentication functionality by automatic transmission of the 6-digit indication from the device via Bluetooth LE.

The transmission between the device and the mobile application was not encrypted, and possible to intercept passively. Below is a clear-text token value as seen in passively intercepted packets decoded in the Wireshark network analyzer:



```
     60 3.978993   unknown_0x582cc410      unknown_0x582cc410  ATT          45 UnknownDir
     61 4.026335   unknown_0x582cc410      unknown_0x582cc410  LE LL        26 Empty PDU
     62 4.074490   unknown_0x582cc410      unknown_0x582cc410  LE LL        26 Empty PDU
     63 4.122612   unknown_0x582cc410      unknown_0x582cc410  LE LL        26 Empty PDU
     64 4.170873   unknown_0x582cc410      unknown_0x582cc410  ATT          35 UnknownDir
     65 4.218882   unknown_0x582cc410      unknown_0x582cc410  LE LL        26 Empty PDU
     66 4.266966   unknown_0x582cc410      unknown_0x582cc410  LE LL        26 Empty PDU
     67 4.315130   unknown_0x582cc410      unknown_0x582cc410  LE LL        26 Empty PDU
     68 4.363130   unknown_0x582cc410      unknown_0x582cc410  LE LL        26 Empty PDU

0000  03 06 26 01 06 2a 06 0a  03 1b 51 3e 00 1d be 00   ..&..*.. ..Q>....
0010  00 10 c4 2c 58 06 13 0f  00 04 00 1d 18 00 00 02   ....X... ........
0020  01 00 06 31 37 38 33 39  34 00 29 e2 fc            ...17839 4.)..
```

4.3. Active interception

Active interception of unencrypted Bluetooth connection is possible when an attacker invokes connections with the device and the mobile application, and relays the messages between them. The devices are led to interpret that they are talking directly to one another, while in fact the transmission is controlled by the attacker. Such an attack is commonly known as “Man in the middle” (MiTM). In such scenarios the attacker can wiretap, alter or inject data into the transmission.

A “proof of concept” of the active attack was implemented in the presented tool. It “clones” the original device for the victim's mobile application. Using the before mentioned strategy (keeping the connection with the original device, and advertising more frequently), it ensures the victim connects to it instead of the device. Next, it can forward and tamper exchanged data, acting as an intercepting "proxy".

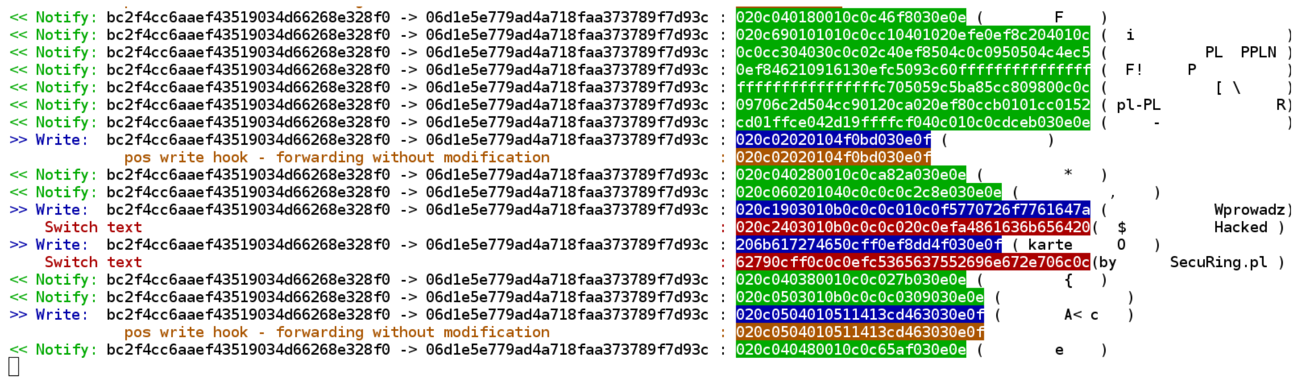
While cloning the MAC address of the original device it is important to "clone" the device services and characteristics along with the exact matching handle numbers. Otherwise, it will not match the mobile OS GATT cache, and the mobile application will not be able to properly communicate. See also chapter 2.5.

The ability to modify and inject the data exchanged between devices can result in various attack possibilities. The attacks depend on the form of data and how it is being transferred, as well as what reactions will be performed by the device or mobile application upon receiving the specific data.

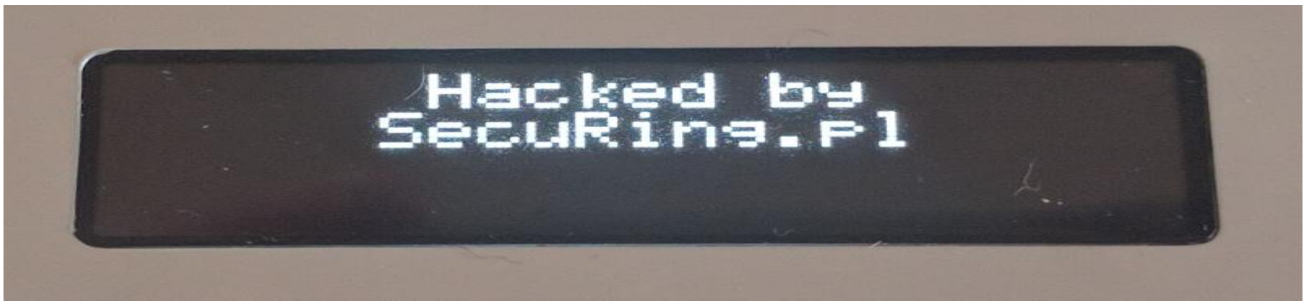
4.3.1. Example vulnerabilities

Data manipulation

An exemplary Point of Sale device was connected to a mobile application via unsecured Bluetooth Smart link. The transaction data was properly encrypted, but the device allowed for a few unprotected commands ("display text" among others). As a result, by switching the original text sent by mobile application and calculating proper CRC (based on algorithm from decompiled Android application), it was possible to display any text on the device during the payment process. The attack did not allow to steal card data, however the weakness could be abused in combination with social-engineering the seller - e.g. by displaying "transaction processed" message on the device after providing invalid PIN.



```
<< Notify: bc2f4cc6aaef43519034d66268e328f0 -> 06d1e5e779ad4a718faa373789f7d93c : 020c040180010c0c46f8030e0e (        F    )
<< Notify: bc2f4cc6aaef43519034d66268e328f0 -> 06d1e5e779ad4a718faa373789f7d93c : 020c690101010c0cc10401020efe0ef8c204010c (  i                  )
<< Notify: bc2f4cc6aaef43519034d66268e328f0 -> 06d1e5e779ad4a718faa373789f7d93c : 0c0cc304030c0c02c40ef8504c0c0950504c4ec5 (            PL  PPLN )
<< Notify: bc2f4cc6aaef43519034d66268e328f0 -> 06d1e5e779ad4a718faa373789f7d93c : 0ef846210916130efc5093c60fffffffffffffff (  F!     P           )
<< Notify: bc2f4cc6aaef43519034d66268e328f0 -> 06d1e5e779ad4a718faa373789f7d93c : ffffffffffffffffc705059c5ba85cc809800c0c (             [ \     )
<< Notify: bc2f4cc6aaef43519034d66268e328f0 -> 06d1e5e779ad4a718faa373789f7d93c : 09706c2d504cc90120ca020ef80ccb0101cc0152 ( pl-PL              R)
<< Notify: bc2f4cc6aaef43519034d66268e328f0 -> 06d1e5e779ad4a718faa373789f7d93c : cd01ffce042d19ffffcf040c010c0cdceb030e0e (     -               )
>> Write:  bc2f4cc6aaef43519034d66268e328f0 -> 06d1e5e779ad4a718faa373789f7d93c : 020c02020104f0bd030e0f (          )
                 pos write hook - forwarding without modification              : 020c02020104f0bd030e0f
<< Notify: bc2f4cc6aaef43519034d66268e328f0 -> 06d1e5e779ad4a718faa373789f7d93c : 020c040280010c0ca82a030e0e (        *   )
<< Notify: bc2f4cc6aaef43519034d66268e328f0 -> 06d1e5e779ad4a718faa373789f7d93c : 020c060201040c0c0c0c2c8e030e0e (        ,    )
>> Write:  bc2f4cc6aaef43519034d66268e328f0 -> 06d1e5e779ad4a718faa373789f7d93c : 020c1903010b0c0c0c010c0f5770726f7761647a (            Wprowadz)
   Switch text                                                                 : 020c2403010b0c0c0c020c0efa4861636b656420(  $         Hacked )
>> Write:  bc2f4cc6aaef43519034d66268e328f0 -> 06d1e5e779ad4a718faa373789f7d93c : 206b617274650cff0ef8dd4f030e0f ( karte    O    )
   Switch text                                                                 : 62790cff0c0c0efc53656375526696e672e706c0c(by      SecuRing.pl )
<< Notify: bc2f4cc6aaef43519034d66268e328f0 -> 06d1e5e779ad4a718faa373789f7d93c : 020c040380010c0c027b030e0e (        {   )
<< Notify: bc2f4cc6aaef43519034d66268e328f0 -> 06d1e5e779ad4a718faa373789f7d93c : 020c0503010b0c0c0c0309030e0e (            )
>> Write:  bc2f4cc6aaef43519034d66268e328f0 -> 06d1e5e779ad4a718faa373789f7d93c : 020c0504010511413cd463030e0f (       A< c   )
                 pos write hook - forwarding without modification              : 020c0504010511413cd463030e0f
<< Notify: bc2f4cc6aaef43519034d66268e328f0 -> 06d1e5e779ad4a718faa373789f7d93c : 020c040480010c0c65af030e0e (       e    )
```



Command injection

For example, a tested car unlocking device implemented its own challenge-response authentication, followed by unencrypted commands and responses exchanged with the mobile application in such authenticated session.

Without altering the authentication process, a “Man-in-the-Midlle” attacker was able to intercept the authenticated session. Next, they actively discarded the original command sent by the mobile application, and instead summoned other ones. The available commands included i.a. overwriting current authentication keys, which could result in taking full control of the affected device.

Additionally, the mobile application service in the background automatically performed authentication in the event of detecting a nearby in-car device, regardless if the proximity auto-unlock feature was on or off. Such behavior makes it easier to attack an unsuspecting victim who is away from device, by simulating the presence of the original device and forwarding packets remotely to it.

Replay

On the contrary to the abovementioned car unlocking device, one example of a smart lock communication protocol did involve encrypting of all the authenticated commands. However, the mechanism did not include protection against replay of encrypted packets. During the challenge-response authentication process, every time the mobile application calculated the same session encryption key in response to a given challenge value.

During the first step of the attack, the intruder could wiretap the challenge-response authentication process and thereafter encrypted communication. Next, during the authentication process, by posing as the original device, the attacker could serve to the mobile application a previously recorded challenge value. Thus the application would calculate the session encryption key matching the wiretapped one. After this point, the attacker could replay the recorded encrypted device responses, and the mobile application would properly decrypt them using the same key.

It the case of the smart lock, the intruder was able to mislead the user, who invoked the “latch” command, into thinking that the lock was properly latched, when in fact the commands invoked by the user were not delivered to the actual device. The unsuspecting user left the premises convinced the door was locked, while the attacker could enter.

4.3.2. Attack countermeasures

The transmission should be properly encrypted - using Bluetooth link-layer security features, or a higher-layer proprietary protocol. For proper implementation of the encryption see also chapter 4.5.3.

Vulnerabilities in proprietary protocols should be prevented by way of proper design and independent evaluation.

4.4. Attacks on exposed services

If the device offers services possible to access without authentication, they may be abused in various ways by an attacker able to approach the affected device.

4.4.1. Example vulnerabilities

Module's AT interface

A Bluetooth module used in an exemplary device implemented a vendor's service, which allowed direct connection to a module's serial AT interface using GATT write/notify requests to predefined characteristics. The interface was not protected. As a result, an unauthenticated attacker could freely change the Bluetooth module’s configuration. According to manufacturer's documentation, it could disrupt the device’s functionality, and probably also damage it physically.

A dedicated module was implemented in the GATTacker tool, and allowed identification of such service in affected devices, detect whether the service is locked, and invoke AT commands to it.

Brute-force

One exemplary device did not implement its soft-lock feature in response to brute-force password guessing. As a result, an attacker could guess the 6-digit password that was protecting access to device for a finite amount of time.

Improper random number generator

Some modules embedded in devices do not provide built-in random number generators. In order to generate random data, developers may use available inputs, which are not sufficiently random. An example solution was to use the current temperature input multiplied by the device's serial number [9].

In many cases the level of randomness has a critical impact on security. For example, in a challenge-response authentication process, where the device generates a random challenge and the mobile application responds with a password-encrypted response. If the challenge value was predictable, an active MITM attacker could emulate the device and trick the mobile application to calculate the proper response for the given challenge. Next, the attacker could use the response to authenticate the actual device.

Excessive services available without authentication

The device may implement excessive services, which are not properly protected. As a result, the unauthenticated attacker may access data or configuration options not intended to be publicly available.

Fuzzing

Sending improper values to characteristics may cause abnormal device behavior.

Logic flaws

Depending on the device, it may be possible to abuse various scenarios, e.g. authentication or access control bypass. An example device stored several authentication keys in the internal module’s register. During authentication, the mobile application indicated which key is used. An attacker could use out of scope key indicator values, which - depending on device’s logic - could be initialized with predictable values. In this way, the attacker was able to bypass authentication.

4.4.2. Attack countermeasures

Definitely inspect all the exposed services before shipping the device to production. Not only restrict access according to the principle of least privilege, but also carefully validate all inputs and prevent logic flaws.

For some devices, a time-limited provisioning may be an acceptable way to prevent misuse of exposed services. For example, devices may expose the configuration services only for a limited time after powering-up or pressing dedicated hardware button.

4.5. Attacks on pairing

The device may implement protected characteristics, which require the connection to be encrypted. Before reading or writing to such a characteristic, devices need to undergo a pairing process and calculate the Long Term Keys which will protect consecutive connections.

Depending on how the devices are paired, it may still be possible to attack such a connection by abusing weaknesses in the implementation and social-engineering users.

4.5.1. “Just Works”

Probably most popular pairing method - “Just Works” - often does not require invoking any action on the device in order to perform a new pairing. In such case, the attacker can create a new bond with a device by simply approaching it and trying to access the protected characteristic. While staying connected to original device, the attacker can then create its “clone”, and trick the victim's mobile application to connect. If the mobile application does not verify the MAC address of the device, the attacker can use its own MAC address for this purpose. The mobile OS checks the current pairing status based on the MAC address of the other device, and in this case the victim's smartphone will not find any pairing information with the attacker's MAC. In effect, it will connect to it without encryption. The attacker can also expose the cloned services without protection, as he does not need to enforce bonding with the victim.

If the attacked mobile application verifies the MAC address of the device, the attacker has to clone it. As a result, the mobile OS will not establish the encrypted connection with the attacker, because the attacker does not know the Long Term Key used for the encryption. In most cases, the mobile application will not display any warning regarding a possible MITM attack, and the users will notice only that they cannot connect to their device. In effect, the disoriented user will probably discard pairing their smartphone, and start the procedure again. Unfortunately, this time they will pair with the attacker, who - from now on - will be able to intercept the traffic.

4.5.2. PIN-protected pairing

With PIN-protected pairing in place, the attacker will not be able to automatically pair with the device. However, they can trick the user into re-initiation of the pairing. As with the situation described above, he can clone the device along with its MAC address. The mobile OS will not be able to establish a secure connection with such a device as the keys will not match. And the user will probably try to remove the pairing and enforce it again. After the user invalidates the pairing, the attacker can switch off their active “cloned” device, and allow the user to continue pairing with the original device. Instead of active interception, they can passively sniff the pairing process. Next, they can crack the PIN and recover the Long Term Key using the Crackle tool [6].

Knowing the Long Term Key, they will be able to proceed with active interception.

4.5.3. Attack countermeasures

The strongest available pairing method should be used, and all the characteristics protected.

Allow pairing initiation only after performing the required action on device - e.g. push a dedicated ‘factory reset’ button.

The mobile application should detect attempts of active interception, and appropriately warn the user. Such functionality may be partly served by mobile OS.

4.6. Whitelisting bypass

Whitelist filtering is based on the MAC address of the accepted device. An attacker may bypass the filtering by changing their MAC address to the whitelisted one.

4.7. Privacy considerations

The publicly available devices's advertisements can be collected and matched against specific individual [11]. Using introduced tool, the collected data can be expanded on services and characteristic values possible to read from the device.

5. ATTACK CONDITIONS, RISK CONSIDERATION

5.1. Physical range

As the Bluetooth operating range is limited, in order to perform a "Man-in-the-middle" attack, an attacker has to be close to both of the attacked devices. The devices do not need to be close to each other, as the attacker can relay packets remotely via an Internet connection. Modular design of the GATTacker tool allows the exploit of such an attack scenario.

Some mobile applications have proximity features which, when improperly implemented, may be abused by approaching the smartphone running the affected application away from the device and its original location.

Also, devices may have vulnerabilities which are possible to exploit directly, without the need to interact with mobile application or intercept the transmission. In such a case, the attacker needs to approach only the vulnerable device.

Mobile malware may attack the BLE devices in range of the infected smartphone. Such malware is operated remotely, and the attack is theoretically possible on a mass-scale.

5.2. Risk

The risk depends on many factors, including the device, its usage and targeted individual.

For example, the current pulse count from a smart-wristband of a regular person is not of much interest to other people. However, the situation may change dramatically if the person is a highly ranked official, and an adversary would like to know their pulse during an important negotiation. Or - the wristband pulse indication is used as a biometric authentication in a banking application.

6. THE NEW TOOL

6.1. Architecture

The tool consists of three main modules:

1. "Central" connecting to original device.
2. "Peripheral" - device emulator.
3. Data interception and manipulation.

The "central" listens for advertisements, scans the device's services for cloning in "peripheral", and forwards the read/write/notification messages exchanged during active attack.

The "peripheral" module loads device specification (advertisement, services, characteristics, descriptors) collected by "central" module, and acts as the device "emulator". It allows to "clone" MAC address of the original device, what is necessary to successfully intercept communication of many mobile applications, which verify the MAC. In such case, the attribute's handle numbers, by which the devices exchange GATT data, must match exactly the original device's ones. Otherwise the mobile OS's GATT cache will not match and prevent the communication.

Data interception and manipulation is possible using hook functions configured via JSON-formatted device . A few example hook functions sources are included in the tool.

The modules can be run on the same system (with at least two Bluetooth 4 interfaces), or on separate ones. They connect to each other using websockets. Thanks to this approach, it is possible to chain the communication – for example to manipulate the BLE requests as JSON text in a web intercepting proxy. It is also possible to invoke remote attacks – where the "central" module is placed near the attacked device, and "peripheral" module close to the victim's smartphone – which can be away from the original device's location.

6.2. Implementation

The tool is written in JavaScript for Node.js framework, using noble [12] and bleno [13] BLE modules by Sandeep Mistry. A bundled version is available as an npm package [14]. The open source code is available on Github [15] under an MIT license.

6.3. Necessary hardware

Each module ("central", "peripheral") requires a Bluetooth Low Energy adapter. The most popular, CSR 8510-based USB dongle is available for about $10, and is confirmed with stable MAC address changing using the Bluez bdaddr tool.

The software is available for Linux systems, and was written in node.js. It was tested on Raspberry Pi.

6.4. Device communication analysis

6.4.1. Mobile application analysis

Mobile application decompilation and code analysis can be very helpful for understanding the communication with device. Application debug log may additionally speed-up the process.

6.4.2. HCI dump

A passive analysis of data exchanged between mobile application and peripheral device can also be performed using "Bluetooth HCI snoop log" Developer Options feature in Android phone. It stores the Host Card Interface dump file in /sdcard/btsnoop_hci.log. The file can later be inspected using Wireshark packet analyzer.

7. REFERENCES

[1] "Generic Access Profile assigned numbers," Bluetooth SIG, [Online]. Available: https://www.bluetooth.org/en-us/specification/assigned-numbers/generic-access-profile.

[2] „Bluetooth GATT specification," Bluetooth SIG, [Online]. Available: https://www.bluetooth.com/specifications/generic-attributes-overview.

[3] „Bluetooth GATT Services specification," Bluetooth SIG, [Online]. Available: https://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx.

[4] „GATT descriptors specification," Bluetooth SIG, [Online]. Available: https://developer.bluetooth.org/gatt/descriptors/Pages/DescriptorsHomePage.aspx.

[5] „Bluetooth Smart Security," Bluetooth SIG, [Online]. Available: https://developer.bluetooth.org/TechnologyOverview/Pages/LE-Security.aspx.

[6] M. Ryan, „Crackle - cracking Bluetooth Smart encryption," [Online]. Available: http://lacklustre.net/projects/crackle/.

[7] „Ubertooth One," Great Scott Gadgets, [Online]. Available: https://greatscottgadgets.com/ubertoothone/.

[8] „Bluefruit LE sniffer," Adafruit, [Online]. Available: https://www.adafruit.com/product/2269.

[9] „Bluetooth Smart community forums: random function," BlueGiga, [Online]. Available: https://bluegiga.zendesk.com/entries/59399217-Random-function.

[10] „Web Bluetooth," [Online]. Available: http://webbluetoothcg.github.io/web-bluetooth.

[11] Mohamed Imran Jameel, Jeffrey Dungen, Low-Power Wireless Advertising Software Library for Distributed M2M and Contextual IoT http://reelyactive.com/science/reelyActive-IoT2015.pdf

[12] Sandeep Mistry, A Node.js BLE (Bluetooth Low Energy) central module https://github.com/sandeepmistry/noble

[13] Sandeep Mistry, A Node.js module for implementing BLE (Bluetooth Low Energy) peripherals https://github.com/sandeepmistry/bleno

[14] https://www.npmjs.com/package/gattacker,

[15] https://github.com/securing/gattacker

www.securing.pl