Hacking the IoT

Bluetooth

Einführendes Wahlfach-Projekt

Angefertigt an der

Fachhochschule FH Campus Wien

Bachelorstudiengang: Informationstechnologien und Telekommunikation

Vorgelegt von:

Stefan Buschbeck

Stefan Trinko

Sebastian Ukleja

Personenkennzeichen

1610475105

1610475130

1610475014

Vertiefungsrichtung:

IT-Security

Eingereicht am:

18. 06. 2018

Kurzfassung

Die vorliegende Arbeit beschäftigt sich mit der Sicherheit von Bluetooth Low Energy Geräten. Mithilfe einer speziellen Hardware dem Ubertooth One wurde an einigen ausgewählten IoT Geräten ausprobiert welche Daten ausgelesen und welche Attacken durchgeführt werden können. Für die Versuche wurden die mit dem Ubertooth One mitgelieferten Tools verwendet sowie Gatttacker, Btlejuice und Gatttool.

Abkürzungsverzeichnis

HCI	Host Controller Interface
RFCOMM	Radio Frequency Communication
BD_ADDR	Bluetooth Device Address
SRES	Signed Response
ACO	Authenticated Cipherig Offset
BLE	Blutetooth Low Engery
L2CAP	Logical Link Control and Adaptation
IKR	Identity Resolving Key
CSRK	Signature Resolving Key
STK	Short Term Key
OOB	Out of Band
LTR	Long Term Key
MITM	Man-in-the-Middle
GAP	Generic Access Protocol
ATT	Attribute Protocol
GATT	Generic Attributes
DoS	Denial of Service

Inhaltsverzeichnis

1 Bluetooth Überblick

1.1 Geschichte

Bluetooth wurde 1998 in einer Kooperation durch Ericcson, Nokia, IBM, Toshiba und Intel entwickelt. Ziel war es eine Lizenzfreie Technologie für universelle Kabellose Datenübertragung bei mobilen Geräten zu entwickeln welche auf billigen energieeffizienten Chips laufen soll und in der Lage ist im 2,4 GHz Band zu operieren. [01]

1.2 Grundlegendes

Bluetooth teilt seine Spezifikation in 2 Teile. Core für die Grundlegende Funktionalität und Hardware sowie die Profiles die festlegen welche Spezifikationen z.B: für ein Bluetooth Headset gelten. Ein Device darf nur dann das Bluetooth Logo laut Bluetooth SIG tragen wenn es diese Spezifikationen erfüllt um eine verlässliche Kommunikation zwischen unterschiedlichen Devices zu ermöglichen.

Core Spezifikationen:

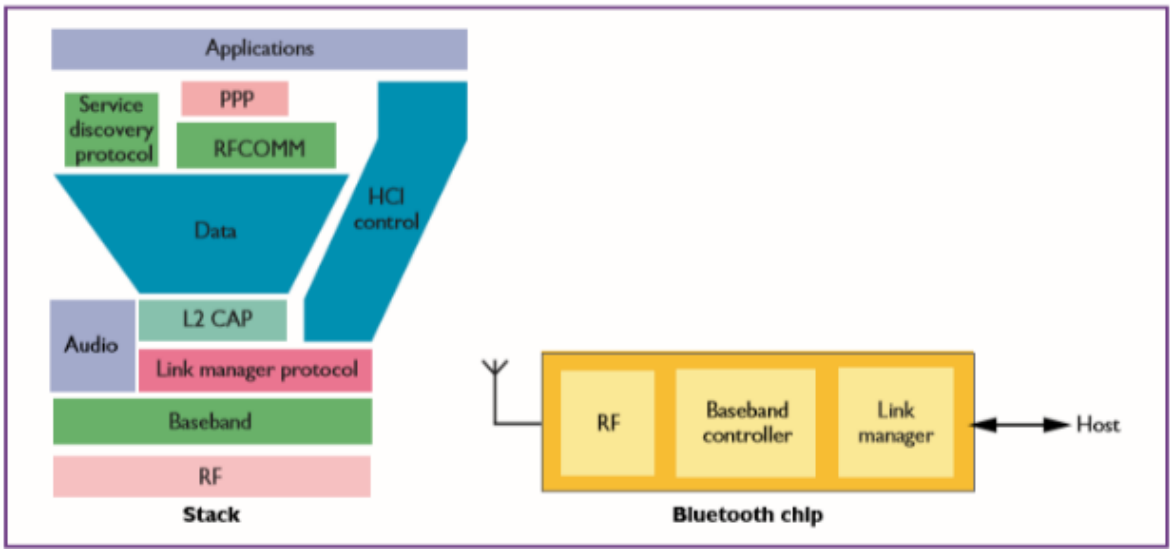


Abbildung 1: Bluetooth Protokoll Stack

Im Protokoll Stack ist das Host Controller Interface (HCI) und das L2CAP besonders hervorzuheben. Der HCI Controller kümmert sich um die Kommunikation zwischen Host Prozessor und Bluetoothchip. Der L2CAP Layer ist praktisch mit Layer 2 aus dem OSI Modell vergleich und kümmert sich um das Identifizieren und Auffinden von Peers. Der Radio Frequency Communication (RFCOMM) in Kombination mit dem PPP Layer kann genutzt werden um IP over Bluetooth zu nutzen. Baseband und RF sind das Äquivalent zum Layer 1 des OSI Modells. [01]

1.2.1 Radio Frequenz

In Europa und Amerika sind für Bluetooth 79 Kanäle im Abstand von 1MHz im 2,4GHz Band spezifiziert. Für Spanien, Frankreich und Japan sind es 23 Kanäle im Abstand von 1MHz. Um resistent gegen Interferenzen zu sein springt Bluetooth durch das ganze Frequenzspektrum und macht dabei 1600 hops pro Sekunde. Die Übertragungsgeschwindigkeit beträgt 1 Mbps.

Die Verbindung und Nutzung einer Hopping Sequenz wird Piconet genannt. Kopfhörer verbunden zu einem Handy würden ein Piconet bilden. Unter normalen Umständen kann ein Master Gerät bis zu 7 Slaves haben. Es sind aber durch Nutzung bestimmter Protokolle auch mehr möglich. Verbindungen zwischen unterschiedlichen Piconets werde durch sogenannte Scatternets realisiert

Ein Host der Mitglied in 2 oder mehr Piconets ist, übernimmt die Rolle eines Bridgenodes und wechselt bei Bedarf auf die Hopping Sequenz des anderen Piconets. [01]

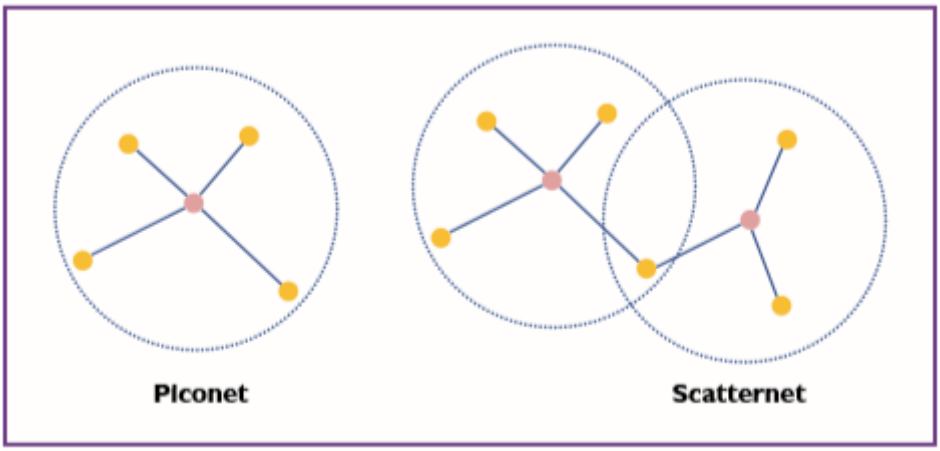


Abbildung 2: Piconet und Scatternet

1.2.2 Verbindungsaufbau

Damit zwei Nodes Daten austauschen können, müssen sie sich auf eine gemeinsame Hopping Sequenz einigen. Um sich überhaupt zu finden wurde eine spezielle Inquiry Hopping Sequenz spezifiziert die alle Bluetooth Devices kennen. Beim Verbindungsaufbau gibt es einen Sender und einen Listener. Der Sender wechselt die Frequenz etwas schneller und sendet eine Nachricht auf jedem Kanal. Damit die Antworten von mehreren Listenern nicht kollidieren haben die einen Backoff Timer mit dem die Antworten leicht verzögert gesendet werden. In diesen Antworten werden die Device Adresse und die Clock offsets transportiert mit denen dann ein Verbindungsaufbau möglich ist.

Sobald die Verbindung aufgebaut wurde erfolgt die Kommunikation nur mehr mit Unicasts und der Listener braucht auch keinen Backoff Timer mehr. [01]

1.2.3 Sicherheit und Authentifizierung

Das Bluetooth Device kann in 3 unterschiedlichen Modi sein:

1. Silent: Das Device akzeptiert keine Verbindungen sondern monitort nur den Traffic.
2. Private: Das Device ist nicht sichtbar und akzeptiert nur Verbindungen wenn es die BD_ADDR (Bluetooth Device Address) kennt.
3. Public: Das Device ist sichtbar und kann von jedem kontaktiert werden.

Die BD_ADDR ist 48 Bit lang. Die ersten 16 Bit bilden den Nonsignifikant Address Part (NAP), die nächsten 8 Bit den Upper Address Part (UAP) und die letzten 24 Bit den Lower Address Part (LAP). NAP und UAP bilden die company_id welche auf den Hersteller verweist. Diese Information ist öffentlich in der IEEE OUI Datenbank ersichtlich ähnlich wie die Identifier für MAC Adressen. Der LAP ist mehr oder weniger ein zufällig generierter Part der Adresse, welcher vom Hersteller selbst gewählt werden kann.

LSB MSB

company assigned						campany id					
LAP						UAP		NAP			
1000	0001	1110	0010	10101	0011	0001	0010	0111	1011	0011	0101

Abbildung 3: Beispiel einer BD_ADDR

Die Verschlüsselung wird bis Version 3.0 mit dem SAFER+ Algorithmus mit 128 Bit realisiert(Ursprünglich ein Kandidat für den AES Algorithmus). Ab Version 4.0 wird der eigentliche AES Standard mit 128 Bit Schlüssellängen verwendet.

Ein Bluetooth Device kann in unterschiedlichen Sicherheitsstufen operieren, aber immer nur in einer gleichzeitig.

1. Nonsecure: Keine Sicherheitsmaßnahmen
2. Sevice-level-enforced security mode: Ein unsicherer Asynchronous Connectionless Link (ACL) wird aufgebaut. Authentifizierung sowie eine optionale Verschlüsselung passiert sobald eine Verbindungsanfrage über das L2CAP Protokoll hereinkommt.
3. Link-level enforced security mode: Die Authentifizierung und Verschlüsselung wird nach dem Aufbau des ACL implementiert.

Um eine Sichere Verbindung aufzubauen muss Bluetooth eine bestimmte Reihenfolge von Abläufen abhalten:

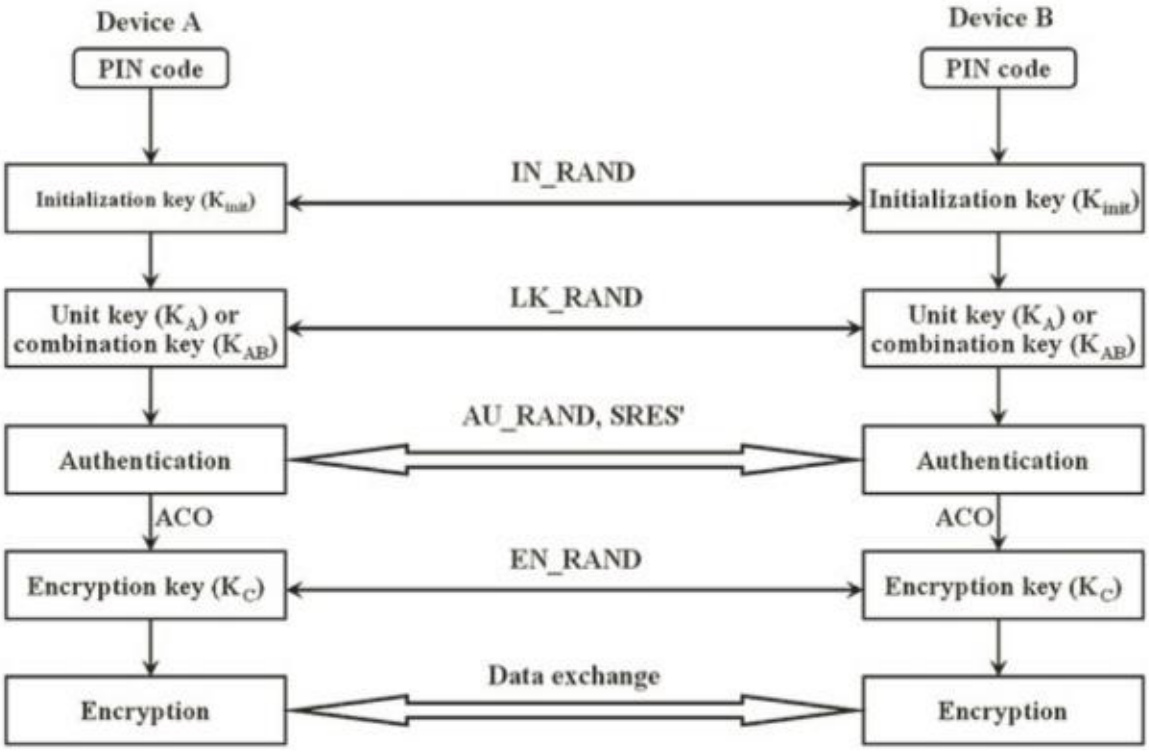


Abbildung 4: Schlüsseltausch

Als erstes wird ein Initializations-Key erstellt sobald sich zwei Bluetooth Devices „treffen“. Der Key wird zusammengesetzt aus einem 128 Bit random Key IN_RAND, einem 1-16 BIT PIN Code und der BD_ADDR eines Device. Der IN_RAND wird unverschlüsselt übertragen.

Die Initialisierung kann als Funktion gesehen werden: Kinit=E(PIN, L, IN_RAND) Der PIN wird hier in den PIN selbst und seine Bitlänge „L“ aufgeteilt. Ist der PIN kleiner als 16 Bit wird er mit der BD_ADDR aufgefüllt bis er 16 Bit erreicht. Falls eine Device einen fixen PIN Code hat, so wird die BD_ADDR des anderen Device genutzt. Wenn beide Devices flexible PIN Codes unterstützen wird die BD_ADDR des Devices genutzt der den IN_RAND Key erhalten hat.

Sobald der Initialization Key „Kinit“ feststeht wird er zum Verschlüsseln einer Zufallszahl RAND oder LK_RAND benutzt. Kinit XOR RAND/LK_RAND wird genutzt um die Zufallszahl zu verschlüsseln.

Die Zufallszahl selbst wird benutzt um einen Link Key (Unit oder Kombination Key) zu erstellen. Ein Unit Key wird aus Informationen von nur einem Device erstellt BD_ADDR und RAND = Ka. Ka wird dann mit Kinit verschlüsselt (Ka XOR Kinit) und an das zweite Device geschickt. Dieses Entschlüsselt mithilfe von (Ka XOR Kinit) XOR Kinit = Ka.

Die Unit Key Variante ist aber sehr unsicher da der Unit Key vom Device für alle Verbindungen genutzt wird. Ein Device kann noch dazu den Traffic aller anderen Devices mit dem selben Unit Key abhören. Zusätzlich kann man sich als ein Device ausgeben indem man einfach die BD_ADDR aus dem Unit Key kopiert.

Die bessere Variante ist es als Link Key einen Combination Key zu verwenden. Dieser Key wird aus den Informationen von beiden Devices gebildet: Kab=(BD_ADDRa, LK_RANDa) XOR (BD_ADDRb, LK_RANDb). Demnach ist es nur eine bitweise XOR Verknüpfung von zwei Unit Keys.

Die Devices müssen nur ihre LK_RAND Nummer austauschen damit sie den Unit Key des jeweilig anderen Devices bauen können. LK_RAND wird zuerst mit Kinit verschlüsselt und ausgetauscht. Wenn z.B.: Device B den verschlüsselten LK_RAND von Device A bekommt (LK_RANDa XOR Kinit) entschlüsselt Device B ihn mit Kinit → (LK_RANDa XOR Kinit) XOR Kinit = LK_RANDa. Damit kann Device B den Unit Key von A bauen, da die BD_ADDR bereits bekannt ist.

Die nächste Phase ist die Challange-Response Authentifizierung. Device A (Der Verifier) sendet eine 128 Bit Zufallszahl AU_RAND an Device B (den Claimant). AU_RAND wird mit dem Link Key (Unit oder Combination) und der BD_ADDR vom Claimant kombiniert. Daraus wird ein 32 Bit SRES (Signed Response) und ein 96 Bit ACO (Authenticated Cipherig Offset) generiert:

Wenn der Verifier feststellt dass beide SRES identisch sind, war die Authetifizierung erfolgreich.

Nun kann die Encryption Phase beginnen um Daten zu verschlüsseln und entschlüsseln. Der ACO, der Link Key und eine neue 128 Bit Zufallszahl

EU_RAND werden genutzt um den Encryption Key zu bilden. Das Master Device (Device A) senden die EU_RAND unverschlüsselt an das Slave Device. Auf beiden Seiten wird der Enkryption Key Kc=(EU_RANDa, ACO, Link Key) gebildet.

Mithilfe des Kc wird dann ein Cipher Stream generiert. Dieser besteht aus Kc, der BD_ADDR des Masters und 26 Bits vom Masters Realtime Clock CLK. Dieser Cipher Stream wird mit den zu sendeten Daten verXORt und an das andere Device geschickt. Diese entschlüsselt die Daten erneut mit XOR von Daten und Cipher Stream. Der Cipher Stream wird mit jedem neuen Baseband Paket neu generiert.

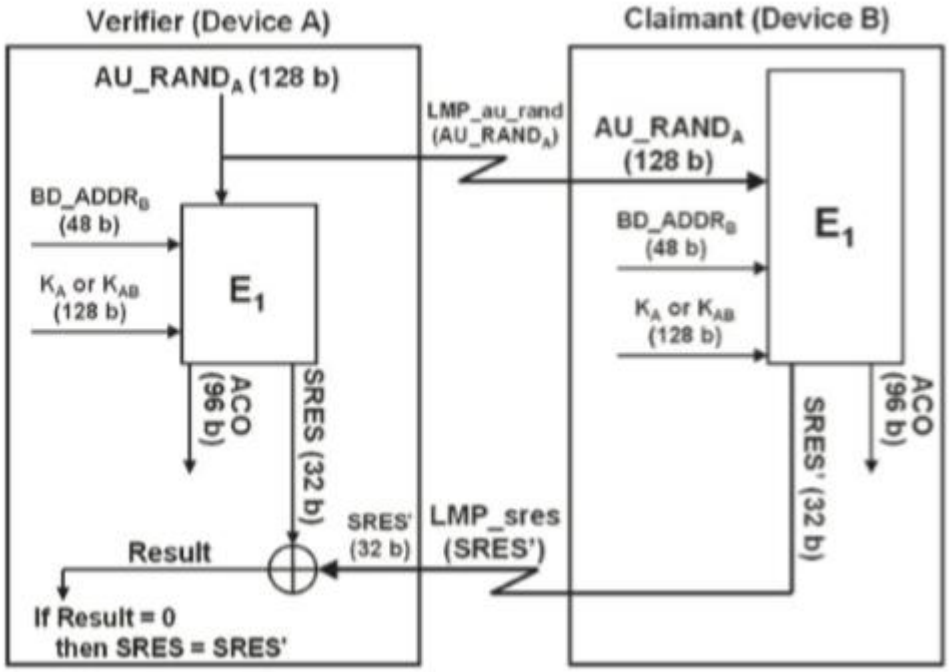


Abbildung 5: Challange Response

Die größte Schachstelle dieses ganzen Ablaufes ist die Tatsache, dass der ganze Prozess nur auf dem PIN basiert (erster Schritt im Prozess). Dieser ist eine schlechte Quelle für Entropie, selbst wenn er aus 16 Stellen besteht (üblicherweise nur 4 Zahlen). Durch den Austausch von Public Keys in dieser Variante sind Man in the Middle Angriffe sehr leicht. Ein Angreifer kann beim Key Austausch die jeweiligen Keys abfangen, sie gegen seine aus tauschen und dann die Kommunikation abhören. Solange keine auffälligen Verzögerungen bei der Kommunikation durch den Key Tausch passieren, ist es unmöglich zu merken ob eine MITM Attacke stattfindet, da die Geräte keine Möglichkeit haben zu verifizieren ob die Keys wirklich von anderen Device stammen.

Um eine MITM Attacke zu erschweren wurde Secure Simple Pairing (SSP) eingeführt. Hier wird mit der Elliptic-Curve-Diffie-Hellmann (ECDH) Kryptographie der Link Key so generiert, dass eine Entropie von ca 95 Bits entsteht. Passives Abhören ist damit mit aktueller Hardware nicht sinnvoll. Zum Pairen nutzt SSP z.B.: NFC oder verlangt von den Benutzern einen 6-stelligen Zahlencode. Sollte beim Pairing durch MITM etwas verändert worden sein, wird es durch SSP zu 99,9% auffallen. [02]

2 Bluetooth Low Energy

1.1 Unterschiede zu Bluetooth

Blutetooth Low Engery (BLE), hat im Gegensatz zu Bluetooth:

- 40 Kanäle zu je 2MHz im 2,4 GHz Bereich, statt 70 zu je 1 MHz
- Einen anderen Protokoll Stack (siehe 2.1.1)
- Gerät geht in den Sleep-Modus wenn es nicht Advertised oder Verbunden ist
- Kann nicht mit Single Mode Classic Bluetooth Devices verbunden werden

1.1.1 Protokoll Stack

<table>
<tr><td colspan="2">Application</td></tr>
<tr><td>GATT</td><td>GAP</td></tr>
<tr><td>ATT</td><td>Security Manager</td></tr>
<tr><td colspan="2">L2CAP</td></tr>
<tr><td colspan="2">Host Controller Interface (HCI)</td></tr>
<tr><td colspan="2">Link Layer (LL)</td></tr>
<tr><td colspan="2">Physical Layer (PHY)</td></tr>
</table>

Abb. 6: Übersicht über den Protokoll Stack

Alles über dem HCI Layer wird als Host Bezeichnet, alles darunter als Controller. Um Host und Controller miteinander zu verbinden, verwendet man das HCI.

- **Physical Layer**:

  Kümmert sich um die Funkverbindung und das Frequenz Hoping. Hierbei sind drei Kanäle sind für das Advertisement gedacht (37, 38 und 39). Alle anderen Kanäle sind Datenkanäle. Alle verbleibenden Kanäle, sind für den Datentransfer vorgesehen.

  Jene Zeit, welche das Gerät in einem Kanal bleibt wird als „Hop Intervall“ bezeichnet. Der Wechsel auf einen anderen Kanal wird durch einen simplen Hop Inkrement zu der aktuellen Kanalnummer berechnet.

- **Link Layer**:

  Übernimmt das Advertising, Scanning sowie den Aufbau und die Verwaltung der Verbindung. Er kann auch Link Layer Encryption liefern. Sendet unter anderem die CONECT_REQ Message um sich mit einem Gerät zu verbinden und die Master Slave Bindung einzugehen.

<table>
<tr><td>Preamble</td><td>Access Address</td><td>Protocol Data Unit (PDU)</td><td>CRC</td></tr>
<tr><td>1 Byte</td><td>4 Byte</td><td>2 - 257 Byte</td><td>3 Byte</td></tr>
</table>

Abb. 7: Link Layer Packet

Durch die Access Address wird eine Verbindung auf einem Kanal identifiziert und ist in der Regel Zufällig, außer im Falle des Advertisements, hier ist die Access Address immer 0x8e89bed6.

Die Encryption auf dem Layer ist eine 128 Bit AES Verschlüsselung. Bevor diese aber stattfinden kann, muss das Pairing passiert sein.

- **L2CAP (Logical Link Control and Adaptation)**:

  Kümmert sich um das Serialisieren von Paketen die von der Applikationsebene kommen, bzw. um das wieder zusammenfügen von Controller seitigen Paketen.

- **Security Manager**:

  Ist für das Pairing zuständig. Es gibt zwei Möglichkeiten das Pairing durchzuführen. LE Legacy Pairing und LE Secure Connections.

  - LE Legacy Pairing:

    Folgende Phasen werden hierbei durchlaufen:

    1. Pairing Feature Exchange:

    Hier geben die Geräte ihre I/O Möglichkeiten bekannt und einigen sich auf eine Verbindungsmethode basiert auf den Möglichkeiten

    2. Short Term Key (STK) Generation:

    Die Geräte generieren einen temporären Key welcher mit einigen Zufallswerten zu einem STK wird. Der STK wird die Übermittlung des Long Term Key absichern.

    3. Key Transport:

    Abhängig von der Einigung in Phase 1 werden die Long Term Keys (LTR) übermittelt. Der LTR selbst ist für die Link Layer Verschlüsselung zuständig. Der Identity Resolving Key (IKR) ermöglicht es einem Device eine Random BDADDR in die reelle public Adresse umzuwandeln. Der Connection Signature Resolving Key (CSRK) dient zur Signierung von Daten um Authentifizierung in unverschlüsselten Verbindungen zu ermöglichen.

    Grundsätzlich ist es auch möglich, dass die Long Term Keys von den Geräten für eine spätere Verbindung abgespeichert werden. [03]

    LE Legacy pairing kann automatisch (Just Works) passieren wie im Falle von IOT Geräten wie Glühbirnen oder Peripherien. Hier ist dann der TK = 0. Pairing kann auch über einen vom User eingegebenen Pin (welcher meist sechstellig) starten (Passkey). In dem Fall wird der TK aus dem Pin erstellt, hier ist die Entropie für den Key allerdings nicht besonders gut. [02]

Die sicherste Methode ist über Out of Band (OOB) den TK zu übermitteln. Hier kann der Key bis zu 128 Bit groß sein. Diese Methode ist allerdings nur solange die Sicherste, wie der OOB Kanal sicher ist. Beispielsweise über NFC.

- LE Secure Connection:

  Wurde mit BLE 4.2 eingeführt und funktioniert erst ab dieser Version aufwärts. Benutzt zur Verschlüsselung einen Elliptic Curve Diffie-Hellmann Algorithmus und ist damit sicherer als die Legacy Variante.

  Phase 1 und 3 sind Identisch wie bei Legacy Verbindung. In Phase 2 wird jedoch bereits ein LTK generiert der die Verbindung verschlüsselt. Die Phase 3 ist hier optional falls ein IKR oder ein CSRK benötigt wird.

  Unterstützt ebenfalls Just Works, Passkey und OOB Pairing wobei hier alle drei Varianten, durch den Algorithmus, sicherer sind als bei der Legacy Verbindung. Zusätzlich gibt es noch eine 4te Variante. Die Numeric Comparison. Hierbei wird dem User beim Verbindungsaufbau eine sechs stellige Nummer generiert. Beide User müssen die selbe Nummer angezeigt bekommen um die Verbindung als nicht kompromittiert zu wissen. Dies ist nur ein Weg zur Minimierung von MITM Attacken und stellt keine Kryptographische Maßnahme dar.

- **GAP (Generic Access Protocol)**:

  GAP kümmert sich um die Zuweisung der Master/Slave Rolle beim jeweiligen Device.

- **ATT (Attribute Protocol)**:

  Ein Attribut ist die kleinst mögliche Dateneinheit welche von den Devices übermittelt werden kann. Das ATT Protokoll kümmert sich um die Speicherung und Zuordnung dieser Attribute.

  Ein Attribut hat:

  - Handle: Eine 16 Bit Address welche das Attribut identifiziert
  - Type: Merkmal, welches kennzeichnet, was ein Attribut repräsentiert
  - Value: der Inhalt des Attributs (die Daten)
  - Permissions: Ob Read/Write Berechtigungen bestehen

- **GATT (Generic Attributes)**:

  GATT ordnet die Attribute von GATT in Profile, Service und Charakteristiken. GATT hat dabei 2 Rollen: Den Client und den Server, wobei der Client Daten vom Server liest und auf den Server schreibt.

  Charakteristiken sind Vergleichbar mit Funktionen. Im Falle einer Glühbirne könnte das z.B. Heller/Dunkler stellen sein.

  Services fassen Charakteristiken zusammen. Um beim Beispiel mit der Glühbirne zu bleiben, könnten ein Lichtsteuerungsservice die Charakteristiken Heller/Dunkler und An/Aus oder Farbe wechseln haben.

  Profile Fassen wiederum Services zusammen.

  GATT stellt hier ein Sicherheitsrisiko dar da man die Kontrolle über die Charakteristiken übernehmen kann, in dem man auf ihre Values schreibt. [03]

1.2 Allgemeine BLE Sicherheitsrisiken

Die größten Sicherheitsrisiken während des Pairing-Prozesses und bei BLE im allgemeinen ist passive Eavesdropping, MITM-Angriffe und Identity-Tracking.

- passive Eavesdropping: Dies ist der Vorgang, bei dem ein unbefugter dritter, den Datenaustausch zwischen zwei verbundenen Geräten mitließt.

  BLE schützt sich gegen Eavesdropping durch die hopping Frequenz. Hier wird 1600 mal pro Sekunde die Frequenz gewechselt und somit das Abfangen des Datenverkehrs erschwert. Die Lücke hier ist simpel: sobald man die BDADDR und die Clock des Master Gerätes hat, kann die hopping Frequenz ermittelt werden. Somit bietet das hopping allein keinen hinreichenden Schutz.[11]

- Man-in-the-Middle-Angriff: Ein MITM Angriff passiert dann, wenn ein drittes Gerät, welches fortan das bösartige Gerät genannt wird, die anderen beiden legitimen Geräte imitiert, um diese Geräte dazu zu bringen, sich mit ihm zu verbinden. Dies Vorgehen gibt den legitimen Geräten die Illusion, dass sie direkt miteinander verbunden sind, wenn auch tatsächlich ihre Verbindung kompromittiert wurde. Diese Einrichtung ermöglicht es dem bösartigen Gerät nicht nur, alle gesendeten Daten abzufangen, sondern auch falsche Daten in die Kommunikation einzufügen oder Daten zu entfernen, bevor sie den beabsichtigten Empfänger erreichen.

  BLE ist sehr anfällig gegen MitM Attacken was naheliegend aufgrund der Funktechnologie und der Tatsache ist, dass die meisten Geräte nur eine Verbindung in ihrem Piconet zulassen. Kapitel 5 demonstriert die MitM Attacke an ausgewählten IoT Geräten.

- Identity-Tracking: Hierbei verknüpft eine Person, die BLE Adresse eines Gerätes mit einem Menschen, welchen die Person verfolgen möchte. Die Verfolgung basiert auf dem Vorhandensein der BLE Adresse oder nicht.

  BLE kann für einen solchen Fall seine Adresse periodisch ändern. [10]

- Denial of Service: Aufgrund der Tatsache, dass die meisten BLE Geräte Batterieabhängig sind, ermöglichen sich hier DoS Angriffsvektoren. In einem Versuch der Universität von Utah, wurde gezeigt wie durch senden einer Vielzahl von Inquiriy Messages an ein Mobiltelefon, seine Batterie innerhalb kürzester Zeit stark reduziert wurde. Das senden von ungütligen Bluetooth Packeten kann an manchen Geräten ebenfalls zu DoS oder unvorhersehbaren Verhalten führen.[11] Kapitel 5 wird demonstrieren wie über eine MitM Attacke ebenfalls ein DoS Effekt herbeigeführt wird.

3 Ubertooth One

Ursprünglich wurde der Ubertooth-Stick als Open-Source-Entwicklungsplattform für Bluetooth-Geräte (also z.B. Fitnesstracker, Smartwatches, Bluetooth-Schlösser und Tastaturen) entwickelt. Damals hatte der Ubertooth-Stick, die Bezeichnung Ubertooth Zero. [05]

Die heutzutage benutze Weiterentwicklung hierfür heißt Ubertooth One. [04]

Mit dem Ubertooth-Stick kann man und macht man auch häufig, die Kommunikation zwischen Bluetooth-Geräten untereinander belauschen, ohne dass man mit ihnen verbunden sein muss. Daher wird man als Mitleser nicht entdeckt.

Da der Stick auch senden kann, lassen sich damit sogar Man-in-the-Middle-Attacken ausführen, z.B., um die Kombination besagter Schlösser zu ändern. Der Ubertooth One kann hierbei sogar mit herkömmlichen Bluetooth, also auch dem häufig in mobilen Geräten anzutreffenden BLE umgehen.

Gelingt es einem Angreifer, den Pairing-Prozess mitzuschneiden, kann er mit einer hoher Wahrscheinlichkeit sogar verschlüsselte Verbindungen knacken. [06]

3.1 Ubertooth One installations Guide

3.1.1 Vorraussetzungen

- Sie betzitzen einen Ubertooth One
- Bluetooth fähiger Computer
- Eine echte Linux-Distribution läuft auf Ihrem Computer. Es spielt dabei keine Rolle ob die Distribution in einer virtuellen Maschiene läuft, oder nicht. Nur der in Windows 10 integrierte Linux-Mode funktioniert nicht.

3.1.2 Installation

Wichtig: Diese Anleitung ist speziell für Debian / Ubunu – Systeme geschrieben.

Um die passenden Befehle für andre Linux -Varianten zu finden, besuchen Sie bitte: https://github.com/greatscottgadgets/ubertooth/wiki/Build-Guide

3.1.2.1. Bluetooth library installieren

```
sudo apt-get install cmake libusb-1.0-0-dev make gcc g++ libbluetooth-dev
\
pkg-config libpcap-dev python-numpy python-pyside python-qt4
```

3.1.2.2. libbtbb installieren (Bluetooth Baseband Library)

Wird von Ubertooth-Tools gebraucht um Bluetooth Packete dekodieren zu können.

```
wget https://github.com/greatscottgadgets/libbtbb/archive/2017-03-
R2.tar.gz -O libbtbb-2017-03-R2.tar.gz
tar xf libbtbb-2017-03-R2.tar.gz
cd libbtbb-2017-03-R2
mkdir build
cd build
cmake ..
make
sudo make install
```

Tipp: Sollten bei Ihnen Fehler auftreten, kann es hilfreich sein „sudo ldconfig" aufzurufen.

3.1.2.3. Ubertooth-Tools

Enthält Host-Code, welcher für das Sniffing von Bluetooth-Paketen benötigt wird, die Konfiguration des Ubertooth und die Aktualisierung der Firmware.

```
wget
https://github.com/greatscottgadgets/ubertooth/releases/download/2017-03-
R2/ubertooth-2017-03-R2.tar.xz -O ubertooth-2017-03-R2.tar.xz
tar xf ubertooth-2017-03-R2.tar.xz
cd ubertooth-2017-03-R2/host
mkdir build
cd build
cmake ..
make
sudo make install
```

3.1.2.4. Firmware aktualisieren

1. Um die aktuellste Firmware auf Ihr Ubertooth-One spielen zu können, stecken Sie Ihr Gerät, an Ihrem Computer an.
2. Wechseln Sie in das Verzeichnis ~/ubertooth-2017-03-R2/ubertooth-one-firmware-bin/
   ```
   cd ~/ubertooth-2017-03-R2/ubertooth-one-firmware-bin/
   ```
3. Ubertragen Sie die neue Firmware
   ```
   ubertooth-dfu -d bluetooth_rxtx.dfu ←
   ```
   Woraufhin die LEDs des Ubertooth kurz zu blinken anfangen sollten
4. Die aktuelle Firmwareversion prüfen
   ```
   ubertooth-util -v
   ```

Nun sollte Ihr Ubertooth-One voll einsatzfähig sein.

Tipp: Sollten bei Ihnen jedoch Fehler auftreten, kann es hilfreich sein „sudo ldconfig" aufzurufen.

Wichtig: Sobald Sie Ihren Ubertooth-One verwenden möchten, muss Bluetooth auf Ihrem Computer eingeschalten sein!

4 Mögliche Zusatzinstallationen

4.1 BtleJuice: The Bluetooth Smart MitM Framework

Es werden hierfür zwei Linux-Maschinen, mit jeweils einem Bluetooth Adapter benötigt

1. Library unter Linux hinzufügen

   ```
   apt-get install bluetooth bluez libbluetooth-dev libudev-dev
   ```

2. Librarys Updaten

   ```
   sudo apt-get update
   ```

3. Node.js und npm installieren

   ```
   apt-get install nodejs
   apt-get install npm
   ```

4. Die eigentliche Installation

   ```
   npm install -g btlejuice
   ```

Eine Maschine übernimmt den Proxy (das Relay). Hier wird nach der Installation nur der Proxy Service gestartet.

```
btlejuice-proxy
```

Auf der Host Maschine (Core) wird die Verbindung zum Proxy aufgebaut:

```
btlejuice -u <Ip-Adresse vom Proxy> -w
```

Danach kann mit einem Webbrowser am Host über localhost:8080 die GUI aufgerufen werden. [07]

4.2 Gattacker

Gattacker ist ein Open Source MitM Tool für BLE, entwichelt von Jasek Slawomir von Securing. Es ist über NPM oder ein Github Repository erhältlich [08].

Das Whitepaper zum Gattacker findet sich auf gattack.io/ [09].

Es werden hierfür zwei Linux-Maschinen, mit jeweils einem Bluetooth Adapter benötigt

1. Library unter Linux hinzufügen
   ```
   apt-get install bluetooth bluez libbluetooth-dev libudev-dev
   ```
2. Librarys Updaten
   ```
   sudo apt-get update
   ```
3. Node.js und npm installieren
   ```
   apt-get install nodejs
   apt-get install npm
   ```
4. Noble installieren
   ```
   npm install noble
   ```
5. Bleno installieren
   ```
   npm install bleno
   ```
6. Die eigentliche Installation von Gattacker
   ```
   npm install gattacker
   ```

4.2.1 Konfiguration

Nach der Installation muss auf beiden Maschinen die **config.env** Datei im **/noble_modules/gattacker** Verzeichnis angepasst werden.

Auf der Host-Maschine:

Hier werden die Einträge **NOBLE_HCI_DEVICE_ID=X** und **BLENO_HCI_DEVICE_ID=X** auskommentiert und Werte statt dem X eingetragen die dem jeweiligen HCI Interface an der Maschine entsprechen (in der Regel 0 oder 1). Falls unklar kann mit **hciconfig** abgefragt werden welche Interfaces benutzt werden.

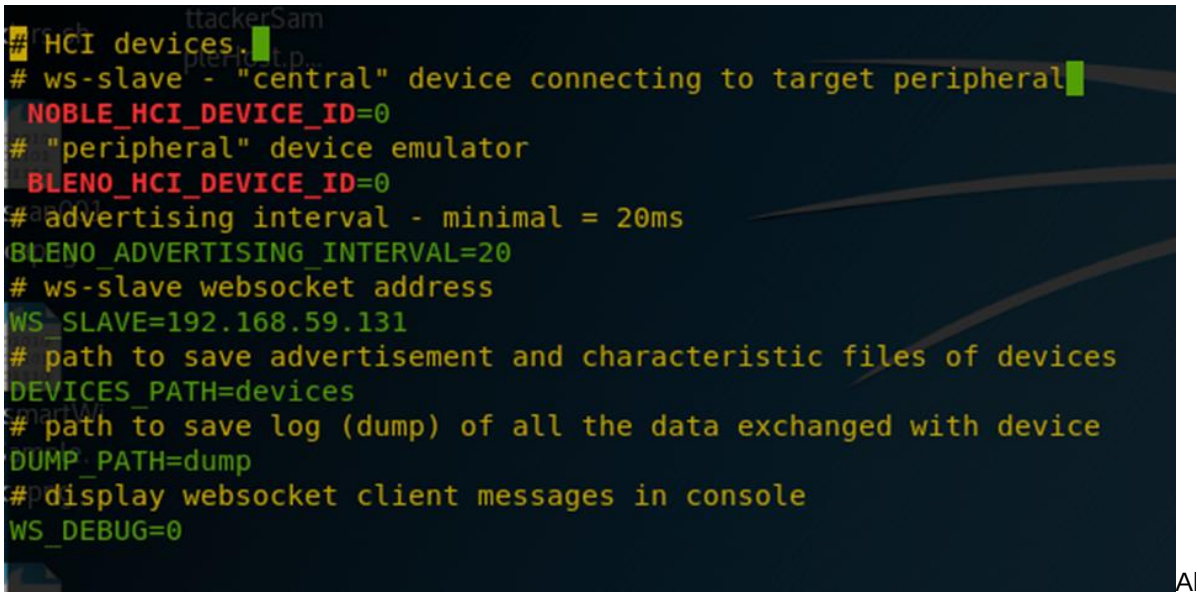


Abbildung 8: Host config.env

Weiters wird die unter **WS_SLAVE=<IP-Adresse>** die IP Adresse der Slave Maschine eingetragen.

Auf der Slave Maschine:

Hier muss nur der **NOBLE_HCI_DEVICE_ID=X** Eintrag auf den richtigen Wert korrigiert und einkommentiert werden:

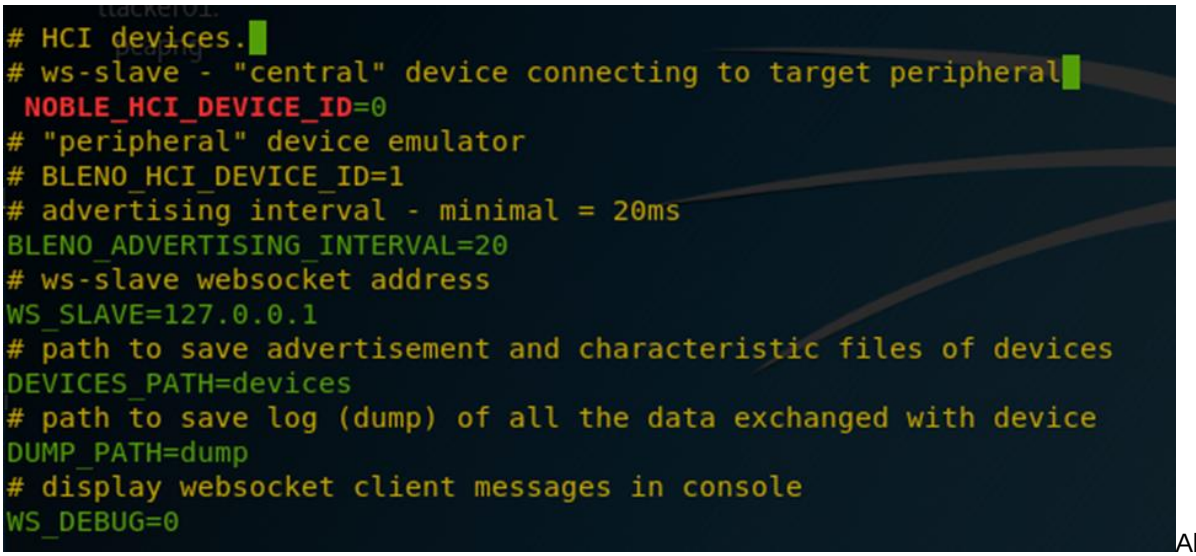


Abbildung 9: Slave config.env

4.2.2 MITM

Auf der Slave Maschine kann jetzt der Proxy gestartet werden. Im /node_modules/gattacker Verzeichnis muss **node ws-slave.js** ausgeführt werden.

Am Host wird nach Advertisments gescannt. Ebenfalls im /node_modules/gattacker Verzeichnis den Befehl **node scan** ausführen.

Jetzt werden die Advertisments vom Gattacker als .json File im Verzeichnis /node_modules/gattacker/devices abgespeichert.

Um sinnvoll spoofen zu können, müssen wir wissen welche Dienste ein BLE Gerät anbietet. Dazu kann uns Gattacker ein Service File generieren. Dies geht mit folgenden Befehl:

node scan <BDADDR>

Mit der Adresse aus dem Advertisement Scan wird Gattacker jetzt eine .srv.json Datei im /devices Ordner erstellen. Gemeinsam mit der Advertisement und der Services Datei kann man das spoofing starten:

node advertise.js -a <Advertisment Datei> -s <Service Datei>

Je nach Rechtevergabe kann es notwendig sein den kompletten Pfad zu den Dateien anzugeben.

Sobald unser Relay mit dem BLE Gerät verbunden ist bekommen wir folgende Meldung:

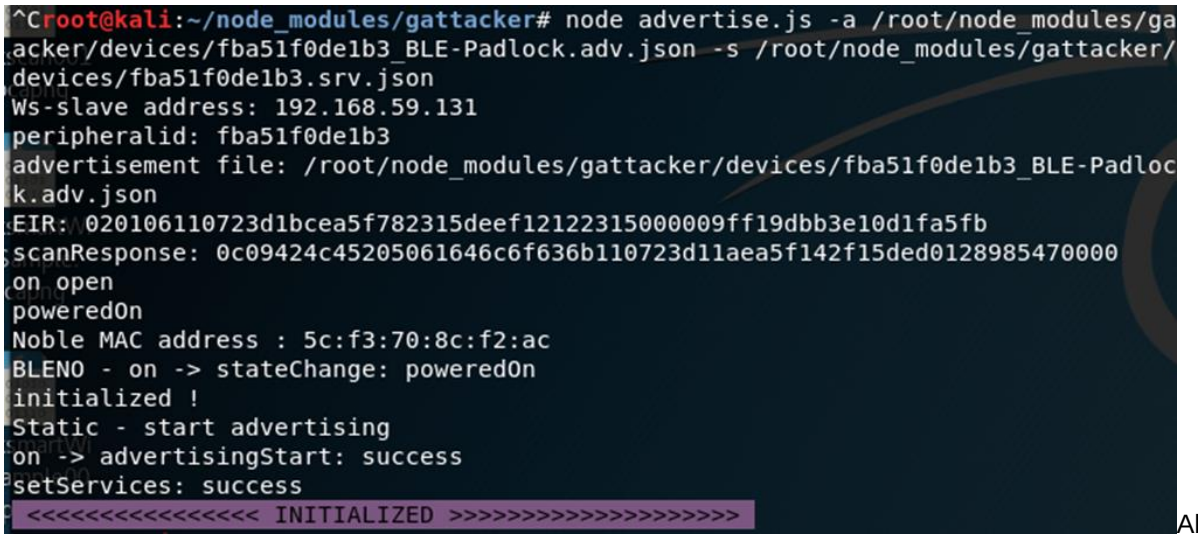


Abbildung 10: Gattacker spoofing

Falls keine weiteren Sicherheitsmaßnahmen vorhanden sind, kann jetzt mit einem Client auf das BLE Gerät zugegriffen werden. Auf der Host Maschine können dann die Read und Write Befehle verfolgt werden.

5 Angriffe mit Ubertooth One

5.1 DogBone Locksmart Mini Keyless Bluetooth Padlock

Beim Locksmart Mini (in weiterer Folge Locksmart genannt) handelt sich um ein schlüsselloses Vorhängeschloss. Gesteuert wird es über eine Mobile APP sowohl für Android als auch IOS Systeme. Die APP erfordert zwingend eine Registrierung mit Email. Weiters verlangt sie einen aktiven Standortdienst. Im Versuch wird das Locksmart mit einem Android Mobiltelefon gesteuert, in weiterer Folge Client genannt.

Das Locksmart ist normal sichtbar und advertised durchgehend:

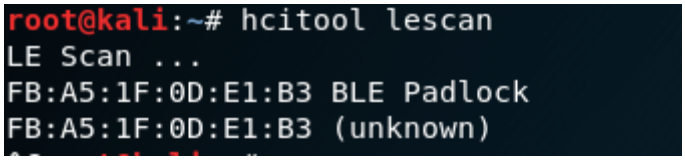


Abbildung 11: HCI BLE Scan Locksmart

Ein Verbindungsaufbau mit dem Locksmart über das Gatttool ist nicht möglich. Die Verbindung wird abgelehnt. Ebenfalls wird eine Verbindung von einem Client ohne aktive Locksmart APP abgelehnt.

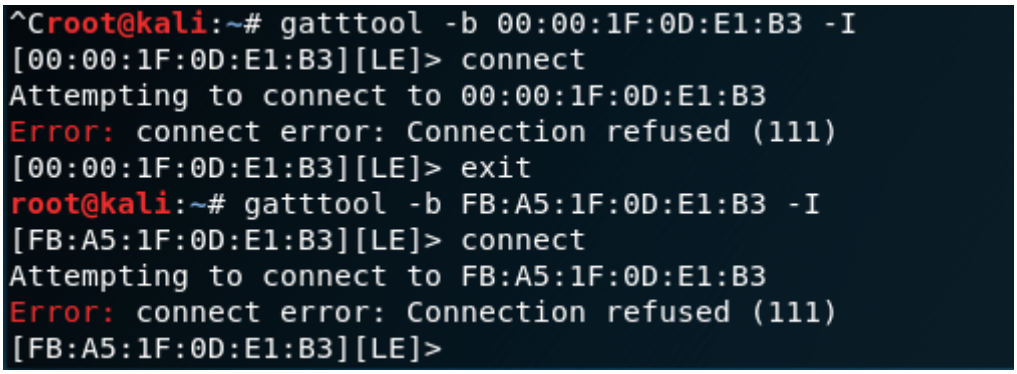


Abbildung 12: Gatttool Locksmart

Über Wireshark können folgende Typen an Nachrichten abgefangen werden: ADV_IND, SCAN_REQ, SCAN_RSP und CONNECT_REQ. Während des Mitschneidens wurde das Locksmart mehrmals mit dem Client geöffnet und geschlossen.

```
root@kali:~# ubertooth-btle -f -tFB:A5:1F:0D:E1:B3 -c /tmp/pipe
systime=1525954613 freq=2402 addr=8e89bed6 delta_t=27.715 ms rss
```

Abbildung 13: Ubertooth Capture Locksmart

Das Locksmart teilt in der ADV_IND seine Seriennummer mit:

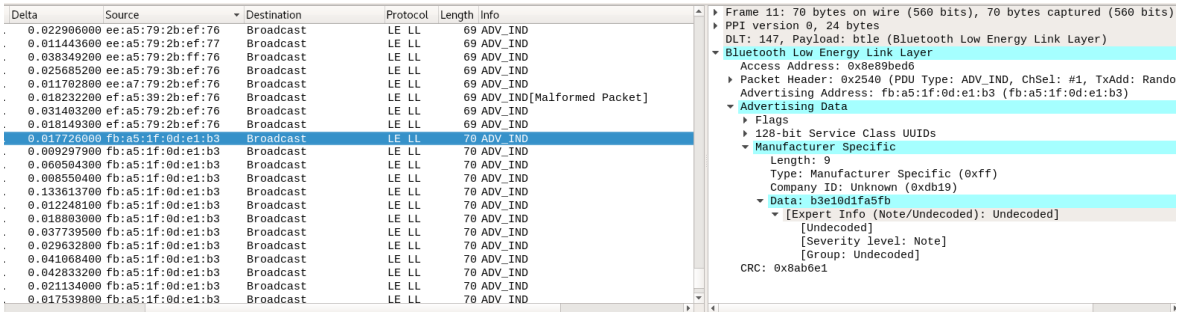


Abbildung 14: Locksmart ADV_IND

Beim CONNECT_REQ können wir nur die vom Client angegeben Daten erkennen. Hier werden die verfügbaren Channels (in dem Fall alle außer 37,38,39) Sowie Informationen zur Clock mitgegeben:

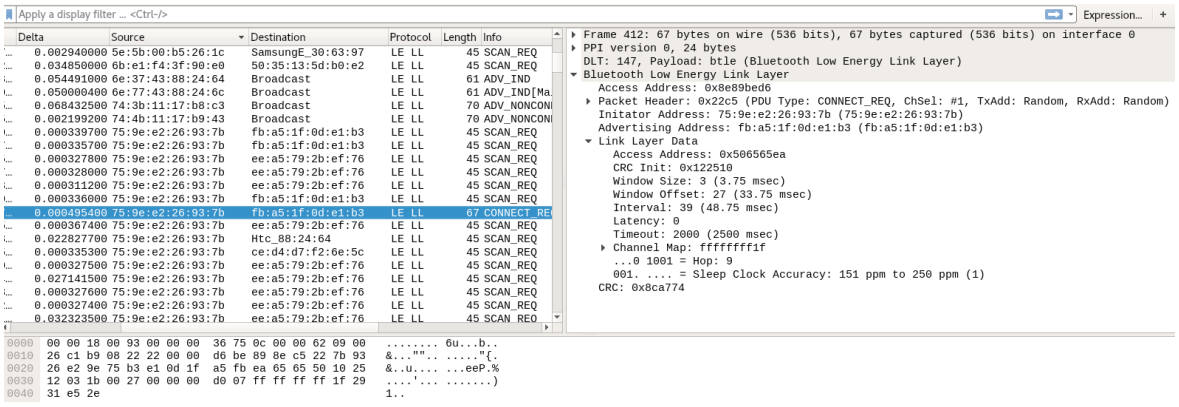


Abbildung 15: Locksmart CONNECT_REQ

Die SCAN_RSP liefert uns keine verwertbaren Informationen für einen Angriff:

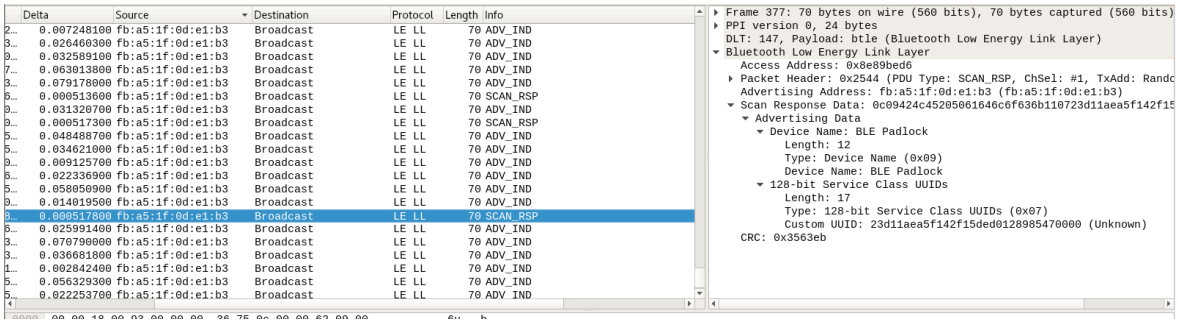


Abbildung 16: Locksmart SCAN_RSP

Alle Nachrichten haben die Access Address 0x8e89bed6. Das bedeutet wir sehen nur Advertisment Information. Mit den oben genutzten Methoden ist es nicht möglich das Pairing oder die Verbindung selbst zu beobachten. Auch können keine ATT oder L2CAP Nachrichten während des Schließens und Öffnens mitgelesen werden. Dies deutet auf eine Verschlüsselung auf Host Ebene hin. Somit wird das Locksmart sowohl Link Layer als auch Host seitige Verschlüsselung bieten, was von einem Schloss auch zu erwarten ist.

5.1.1 Man-in-the-Middle

5.1.1.1. Gattacker:

Die Verbindung zum Smartlock über ein Gattacker Relay ist möglich:

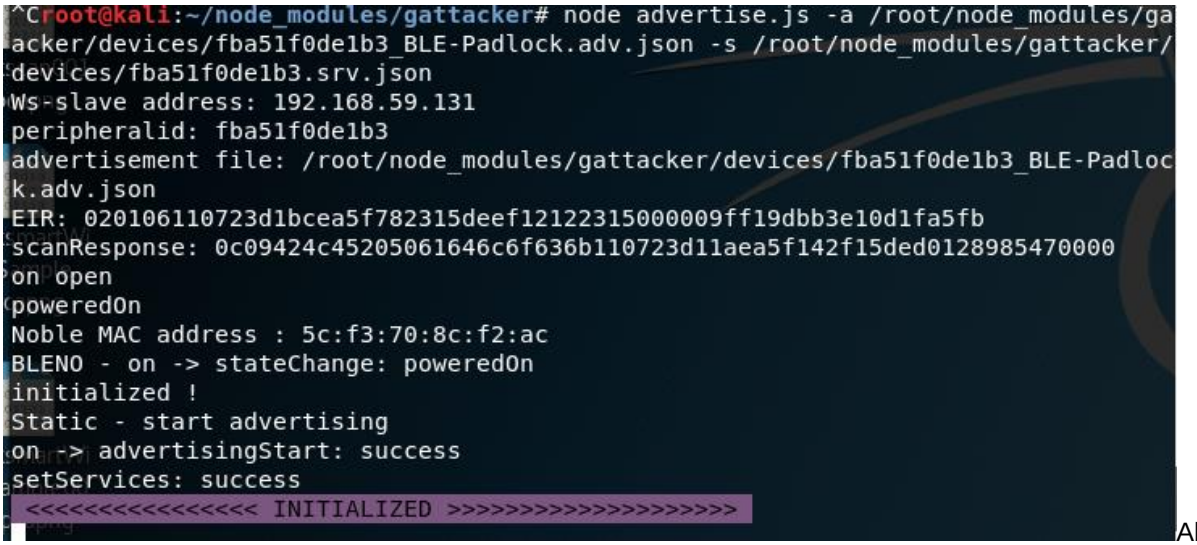


Abbildung 17: Gattacker Locksmart spoofing

Die Verbindung wird initialisiert. Der Client kann sich jedoch nicht mehr über die App zum Locksmart verbinden. Damit wurde ein Denial of Service (DoS) ausgelöst. Weiters ist auffällig, dass beim Client unter den gescannten Bluetooth Geräten zwei Geräte mit dem Namen Padlock auftauchen, was die Attacke sehr offensichtlich macht.

5.1.1.2. Btlejuice:

Auch bei Btlejuice funktioniert der Verbindungsaufbau als Proxy. Die Verbindung wird aber unmittelbar danach abgebrochen:

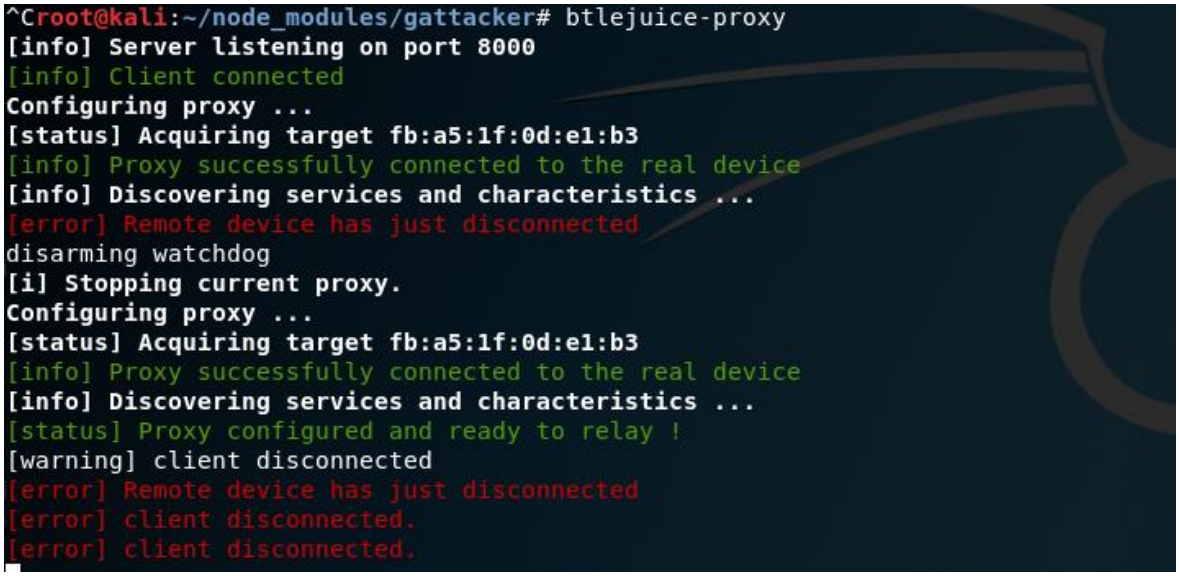


Abbildung 18: Locksmart Btlejuice spoofing

Ein Mitschnitt im Wireshark auf der Host Maschine liefert während des Gattacker Einsatzes zusätzlich einige ATT Nachrichten, die jedoch alle nur MTU Requests und Responses sind.

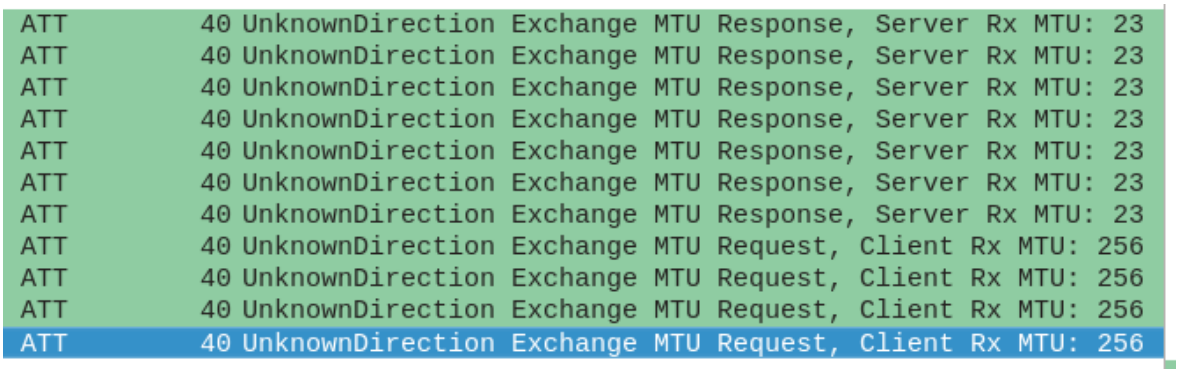


Abbildung 19: Wireshark Locksmart MTU

Ein MTU Request:

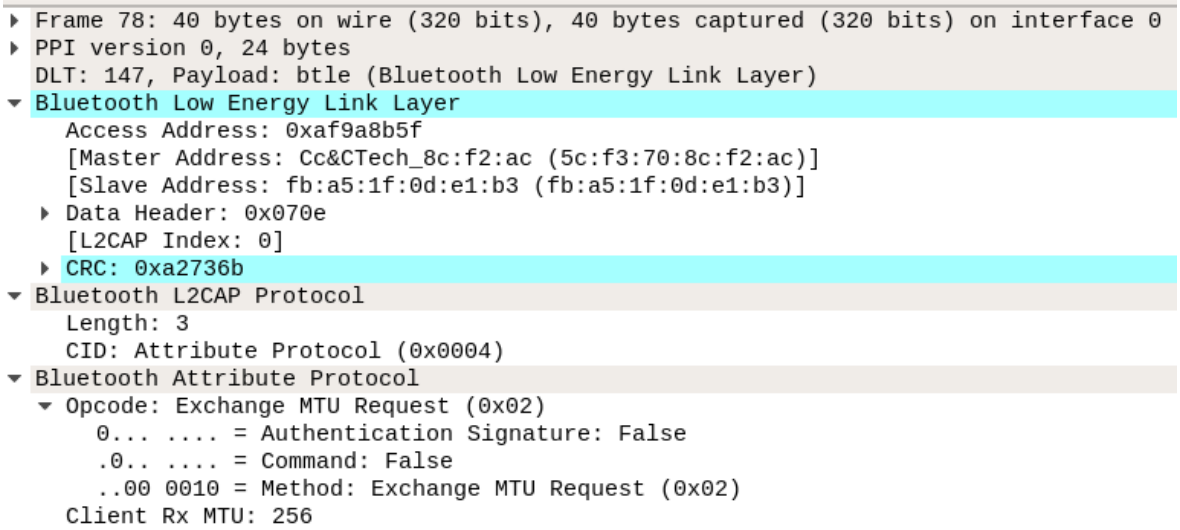


Abbildung 20: Locksmart MTU Request

Ein DOS Angriff ist also über Gattacker möglich, jedoch ist es nicht gelungen im Versuchsaufbau die ATT Nachrichten für das schließen oder öffnen abzufangen.

5.2 Osram Smart+ Classic E27 Multicolor

Die Osram Smart+ (in weiterer Folge Osram genannt) ist eine mehrfarbige LED Glühbirne mit Bluetooth Anbindung. Sie lässt sich über das Apple Homekit steuern. Eine Steuerung der Glühbirne ist nur über das Homekit möglich und steht somit Android Geräten nicht zur Verfügung.

Die Osram advertised öffentlich und schickt als Namen „A60 Ce“ mit:

Abbildung 21: HCI BLE Scan Osram

Ein Zugriff über das Gatttool ist nicht möglich. Die Verbindung wird abgelehnt:

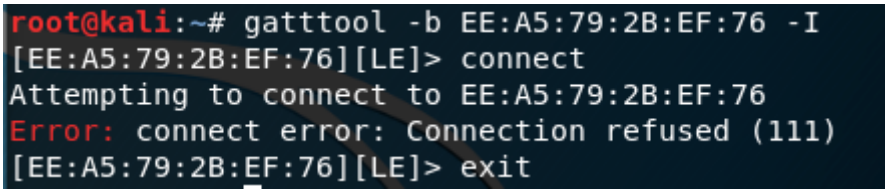


Abbildung 22: Gatttool Osram

Da das Home Kit erst nach einlesen eines 8 stelligen Codes die Verbindung aufnehmen kann, wird die Verbindung darüber authentifiziert.

Im Wireshark können die ADV_IND Nachrichten beobachtet werden:

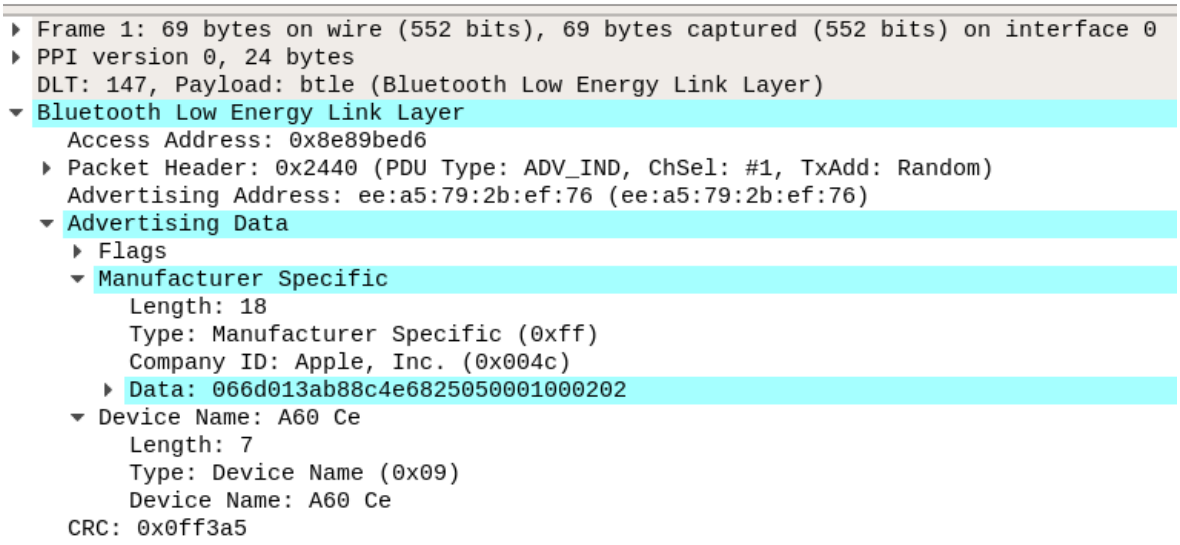


Abbildung 23: Osram ADV_IND

Obwohl die Osram nicht von Apple ist, sendet die Advertisment Nachricht die Company ID von Apple mit, was darauf hinweist, dass der Bluetooth Teil der Osram von Apple ist, oder zumindest die Link Layer Encryption durch Apple bereitgestellt wurde.

Während dieses Mitschnittes wurde keine Interaktion von einem Client auf die Osram durchgeführt.

Sobald die Osram während des Mitschnittes von einem Client mit Home Kit angesprochen wird lässt sich auch die Interaktion im Wireshark beobachten. Bei jedem Ein und Ausschalten der Osram über den Client können wir die ATT Nachrichten beobachten. Es ist auch die L2CAP Nachricht zu sehen in dem ersichtlich ist, dass die Osram die Slave Rolle einnimmt und der Client die Masterrolle:

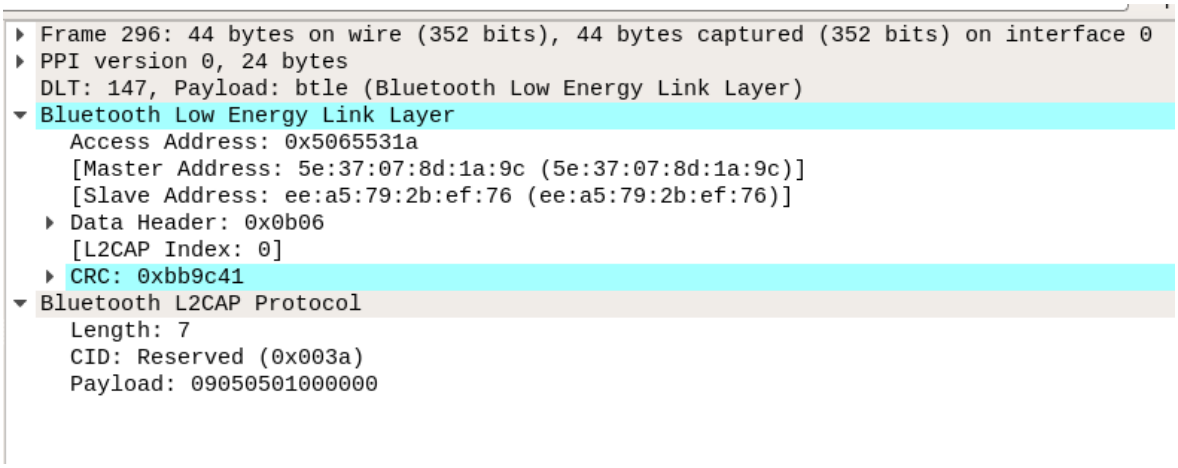


Abbildung 24: Osram LCAP Nachricht

Bei den ATT Nachrichten können die Befehle, welche gesandt werden, sogar erkannt werden.Hier ist ein Write Request zu erkennen:

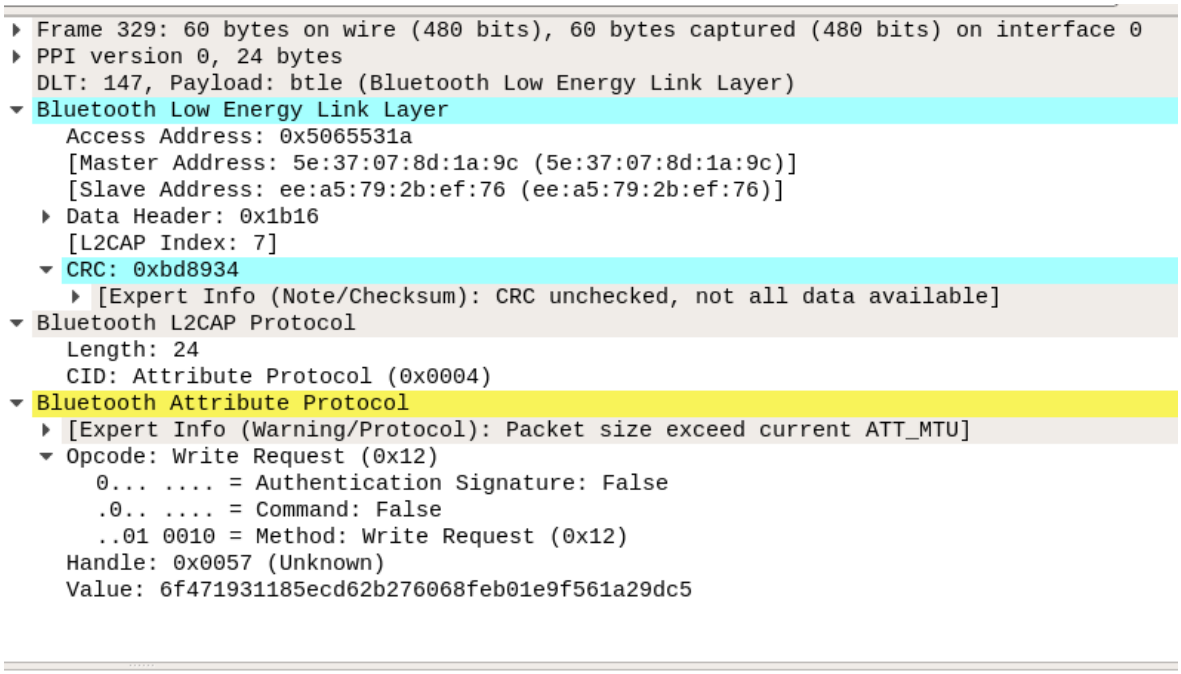
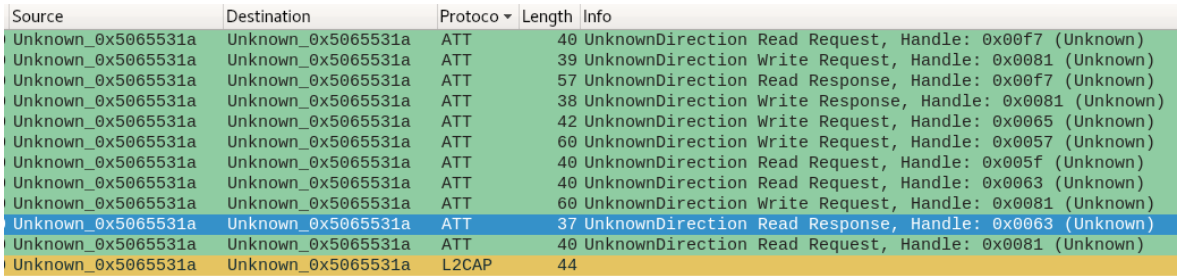


Abbildung 25: Osram Write Request

Der Eintrag „Value“ am Ende des Bluetooth Attribute Protocol gibt den Befehl an der geschrieben werden soll.

Zusätzlich sind noch Write Response, Read Request und Read Response zu sehen:



Source	Destination	Protoco	Length	Info
Unknown_0x5065531a	Unknown_0x5065531a	ATT	40	UnknownDirection Read Request, Handle: 0x00f7 (Unknown)
Unknown_0x5065531a	Unknown_0x5065531a	ATT	39	UnknownDirection Write Request, Handle: 0x0081 (Unknown)
Unknown_0x5065531a	Unknown_0x5065531a	ATT	57	UnknownDirection Read Response, Handle: 0x00f7 (Unknown)
Unknown_0x5065531a	Unknown_0x5065531a	ATT	38	UnknownDirection Write Response, Handle: 0x0081 (Unknown)
Unknown_0x5065531a	Unknown_0x5065531a	ATT	42	UnknownDirection Write Request, Handle: 0x0065 (Unknown)
Unknown_0x5065531a	Unknown_0x5065531a	ATT	60	UnknownDirection Write Request, Handle: 0x0057 (Unknown)
Unknown_0x5065531a	Unknown_0x5065531a	ATT	40	UnknownDirection Read Request, Handle: 0x005f (Unknown)
Unknown_0x5065531a	Unknown_0x5065531a	ATT	40	UnknownDirection Read Request, Handle: 0x0063 (Unknown)
Unknown_0x5065531a	Unknown_0x5065531a	ATT	60	UnknownDirection Write Request, Handle: 0x0081 (Unknown)
Unknown_0x5065531a	Unknown_0x5065531a	ATT	37	UnknownDirection Read Response, Handle: 0x0063 (Unknown)
Unknown_0x5065531a	Unknown_0x5065531a	ATT	40	UnknownDirection Read Request, Handle: 0x0081 (Unknown)
Unknown_0x5065531a	Unknown_0x5065531a	L2CAP	44	

Abbildung 26: Osram ATT Nachrichten

Auffällig ist, dass in beim ATT und L2CAP Protokoll sowohl die Absender als auch die Zieladresse die Access Address des Kanals ist.

Da die ATT Nachrichten mitgelesen werden können ist keine Verschlüsselung auf Host Seite zu erkennen.

Mithilfe der Informationen in den ATT Nachrichten die mit Wireshark über den Ubertooth ausgelesen wurden, können die ATT Attribute verändert werden. Voraussetzung ist jedoch, dass eine Verbindung zum Gerät möglich ist. Die Osram wird jedoch jede Verbindung ablehnen welche nicht von einem Gerät kommt das sich über den mitgelieferten Code zu ihr Verbunden hat.

5.2.1 Man-in-the-Middle

5.2.1.1. Gattacker:

Gattacker kann nicht die benötigte Service Datei anhand der Advertisment Datei anlegen. Aus diesem Grund ist es nicht möglich eine MITM mit Gattacker durch zuführen da wir nicht die BDADDR spoofen können.

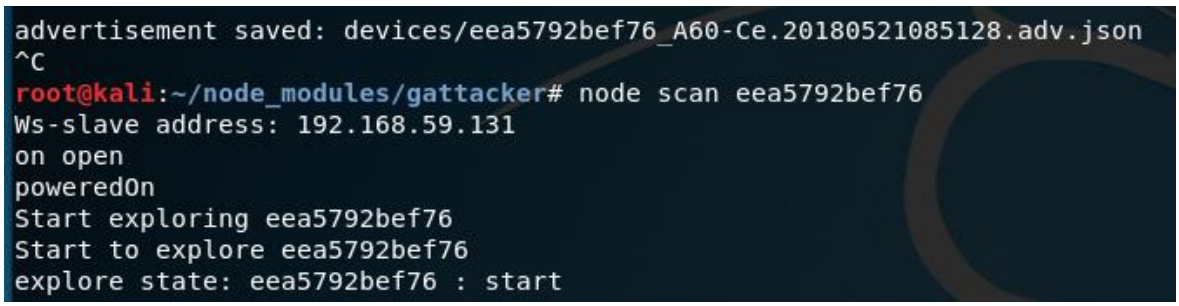


Abbildung 27: Osram Gattacker Scan

5.2.1.2. Btlejuice:

Btlejuice ist in der Lage die Adresse zu spoofen. Sobald aber der Client über das Homekit auf die Osram zugreifen will bricht die Verbindung zum Btlejuice Proxy ab. Hier scheint eine Authentifizierung fehl zuschlagen. Das deutet darauf hin, dass auch nach der Verbindung die Authentifizierung über den mitgelieferten Code erfolgt da unser Proxy nur die BDADDR der Osram spooft können wir uns dennoch nicht bei der Homekit App authentifizieren.

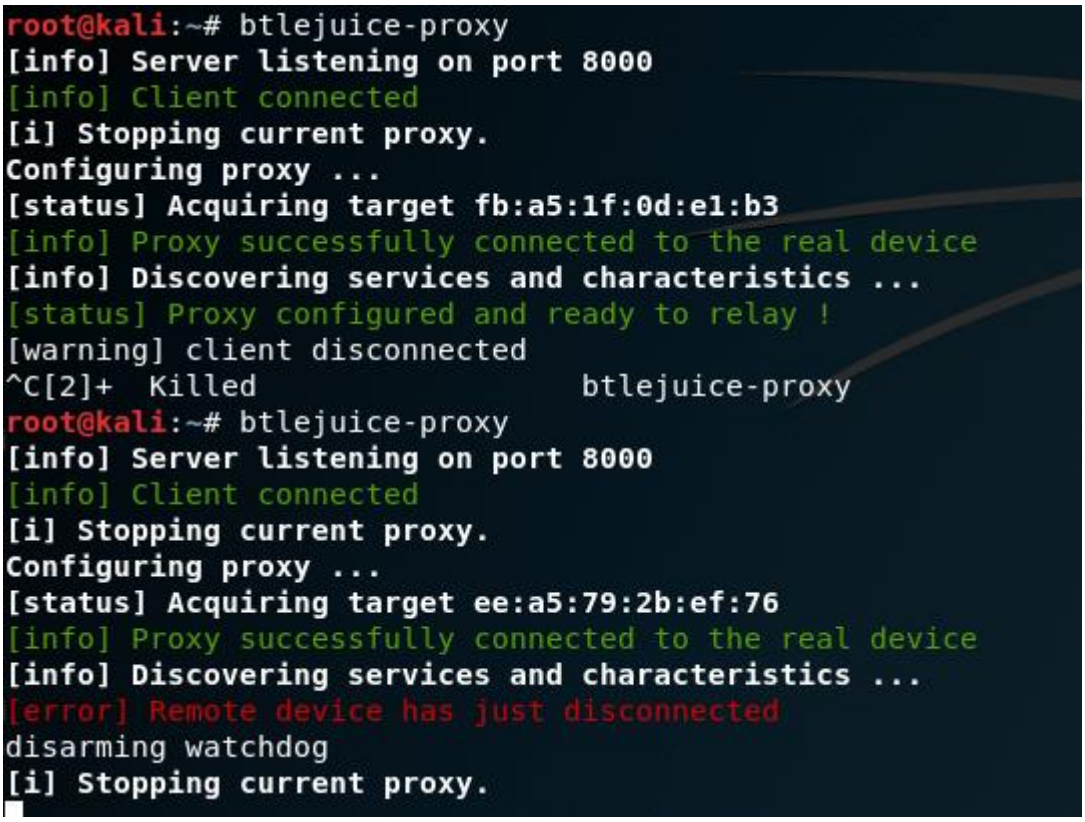


Abbildung 28: Osram Btlejuice

5.3 Equiva Türschlossantrieb

Der Equiva Türschlossantrieb ermöglicht es über eine mobile App sein Türschloss zu bedienen. Die App ist für Android und Apple IOS erhältlich. Über einen QR Code authentifiziert sich die APP beim Türschlossantrieb.

Das Schloss advertiesed, wie schon des Öfteren gesehen, seinen Namen.

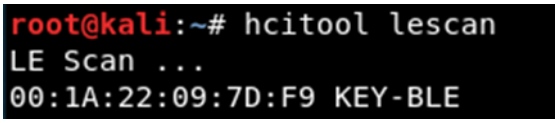


Abbildung 29: Equiva advertising

Eine Verbindung mittels dem Gatttool ist ebenfalls möglich. Die Verbindung wird nur dann hergestellt, wenn gerade kein anderes Gerät mit dem Türschlossantrieb verbunden ist. Weshalb dies auch bei diesem Gerät für eine DoS Attacke nutzbar ist.

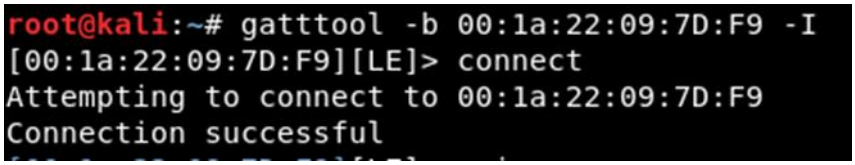


Abbildung 30: Equiva Gatttool Verbindung

Außerdem lassen sich alle verfügbaren Handles auslesen:

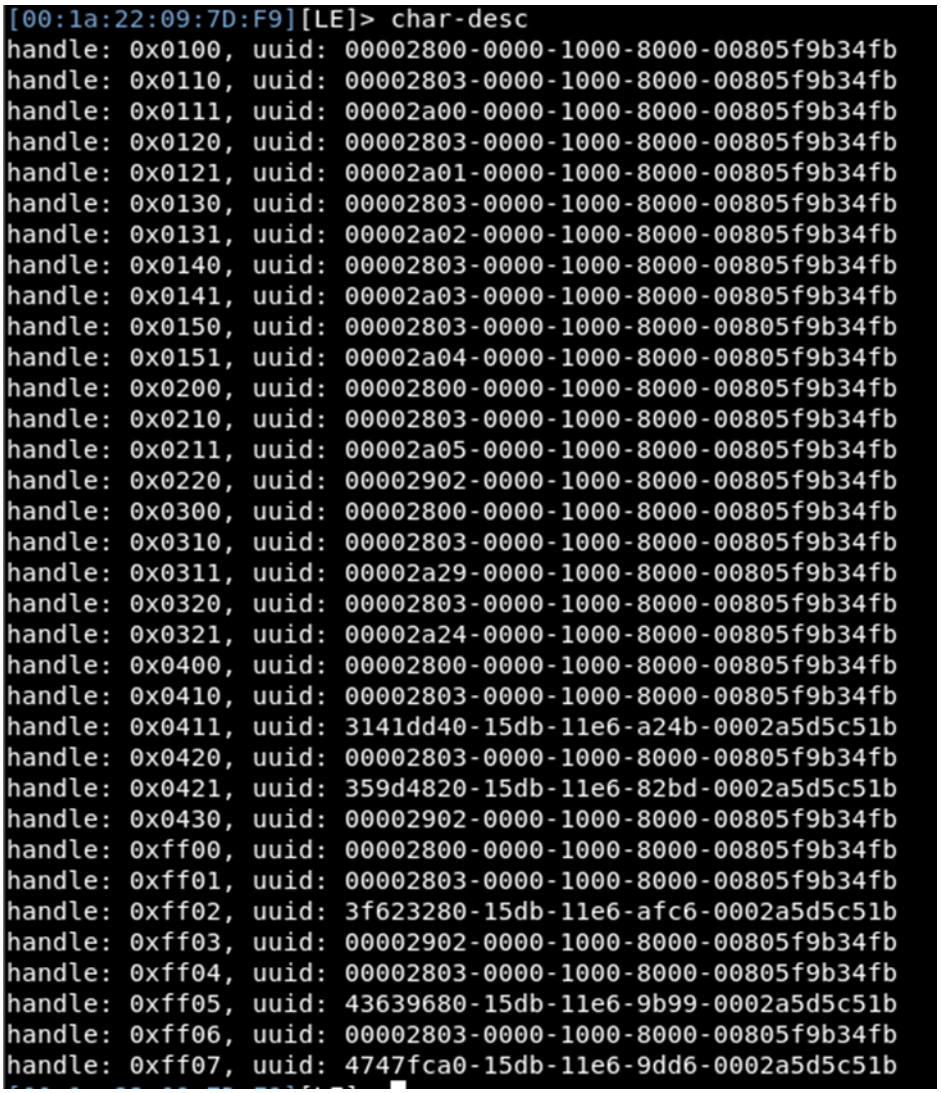


Abbildung 31: Equiva Handles

Das Auslesen der Deskriptoren/Values ist wiederum nur bei wenigen Handles möglich:

```
[00:1a:22:09:7D:F9][LE]> char-read-hnd  0x0132
Characteristic value/descriptor: 00 18
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Characteristic value/descriptor: 02 11 01 00 2a
Characteristic value/descriptor: 4b 45 59 2d 42 4c 45
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Characteristic value/descriptor: 02 21 01 01 2a
Characteristic value/descriptor: 00 00
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Error: Characteristic value/descriptor read failed: Invalid handle
Characteristic value/descriptor: 02 31 01 02 2a
Characteristic value/descriptor: 00
Error: Characteristic value/descriptor read failed: Invalid handle
```

Abbildung 32: Equiva Values

Hier ist nur ein Wert in UTF-8 übersetzbar:

„4b 45 59 2d 42 4c 45“ bedeutet „KEY-BLE“

Wird das Schloss während der Verbindung geöffnet oder geschlossen, so kann mitgesnifft werden, welches Attribut verwendet wird:

```
[00:1a:22:09:7D:F9][LE]> connect
Attempting to connect to 00:1a:22:09:7D:F9
Connection successful
Notification handle = 0x0421 value: 80 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Notification handle = 0x0421 value: 80 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Abbildung 33: Handles beim öffnen und schließen

Das Auslesen der Handle bringt allerdings keine verwertbaren Informationen:

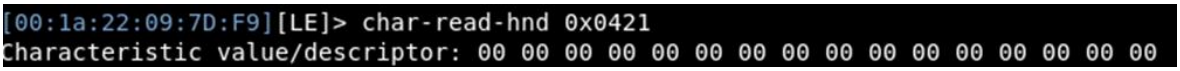


Abbildung 34: Handle 0x421 Value

5.3.1 Man-in-the-Middle

Eine MitM Attacke via Btlejuice schlägt fehl, da gleich nach Verbindungsaufbau durch den Proxy die Verbindung abbricht. Hier wird eine Authentifizierung vermutet.

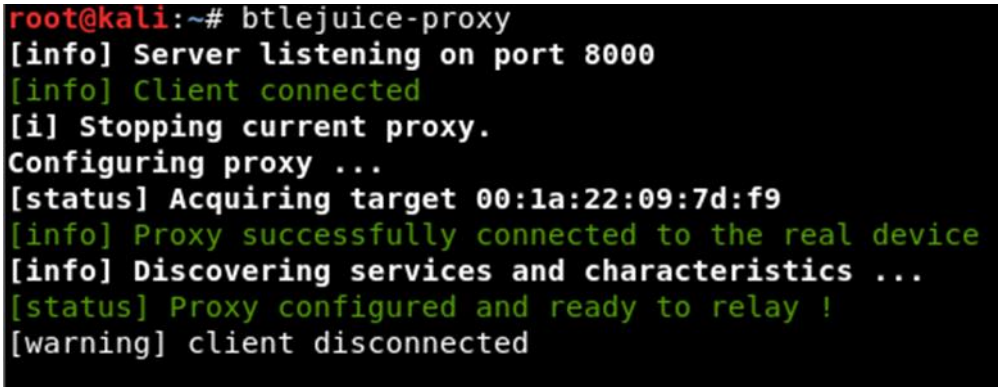


Abbildung 35: Equiva Btlejuice

Mit dem Ubertooth sehen wir nur ADV und SCAN Messages im Wireshark. Die ATT bleiben beim Versuch mitzuschneiden verborgen. Es ist anzunehmen, dass wir im Versuch nicht die richtige hopping Frequenz erwischt haben was bei Einsatz nur eines Ubertooths vorkommen kann.

5.4 BL-5 Lighbulb

Die BL-5 ist eine Bluetoothfähige LED Lampe, welche zusätzlich Musik abspielen kann. Angesteuert wird sie über eine Mobile App, erhältlich für Android und Apple IOS.

Die BL-5 advertised ihren Namen und lässt eine Verbindung über Gatttool zu. Es ist sogar möglich mehrere Verbindungen aufzubauen. Der Traffic der parallelen Verbindung konnte dabei jedoch nicht mittels Gatttool mitgelesen werden. Eine DoS Attacke durch Verbindungsaufbau wie bei den vorherigen Geräten ist somit nicht möglich.

Abbildung 36: BL-5 Gatttool Verbindung und Handles

Es ist auch möglich über den Befehl primary die Handles auszulesen. Man kann im Falle der BL-5 sogar alle verwendeten Attribut Werte auslesen:

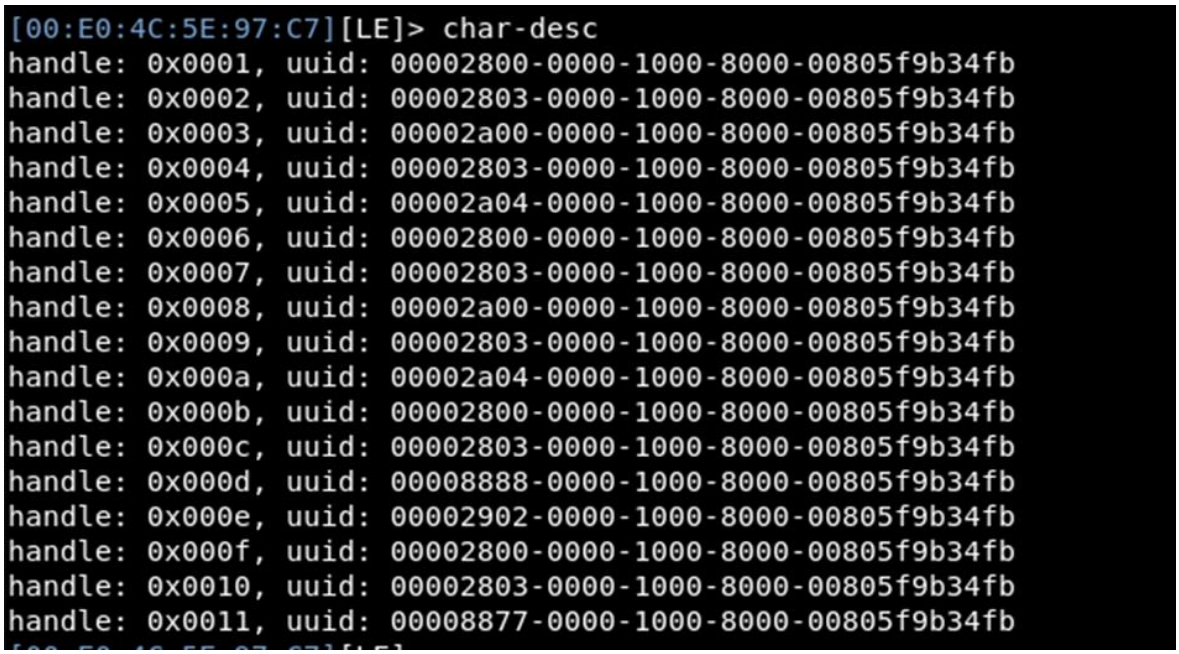


Abbildung 37: BL-5 Handles

Jede dieser Handles kann mit dem Befehl char-read-hnd ausgelesen werden. Gatttool liefert damit alle Werte (Values) der einzelnen Attribute welche wie folgt lauten:

Ausgelsesene Values:

Characteristic value/descriptor: 00 18

Characteristic value/descriptor: 0a 03 00 00 2a

Characteristic value/descriptor: 42 4c 5f 30 35 28 42 4c 45 29

Characteristic value/descriptor: 02 05 00 04 2a

Characteristic value/descriptor: 06 00 06 00 00 00 0a 00

Characteristic value/descriptor: 00 18

Characteristic value/descriptor: 0a 08 00 00 2a

Characteristic value/descriptor: 4c 45 20 53 61 6d 70 6c 65 20 44 65 76 69 63 65 00

Characteristic value/descriptor: 02 0a 00 04 2a

Characteristic value/descriptor: 06 00 06 00 00 00 0a 00

Characteristic value/descriptor: 66 66

Characteristic value/descriptor: ee 0d 00 88 88

Characteristic value/descriptor: 02 00

Characteristic value/descriptor: 77 77

Characteristic value/descriptor: 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77

Characteristic value/descriptor: 77 77 77 77 77 77 77 77 77

Die einzelnen Hex Werte können in eine für den Menschen leicht lesbare Form umgewandelt werden. Allerdings ergeben auch dann nur zwei Werte tatsächlich (menschlichen) Sinn.

4c 45 20 53 61 6d 70 6c 65 20 44 65 76 69 63 65 00 → LE Sample Device

42 4c 5f 30 35 28 42 4c 45 29 → BL_05(BLE)

Alle anderen Werte ergeben in UTF-8 kein sinnvolles Format. Um dennoch herauszufinden, welche Values, welche Eigenschaften an der BL-5 verändern, muss der Bluetooth Traffic mitgesnifft werden.

5.4.1 Man-in-the-Middle

Ein MitM ist uns nicht gelungen. Btlejuice konnte keine Verbindung aufbauen.

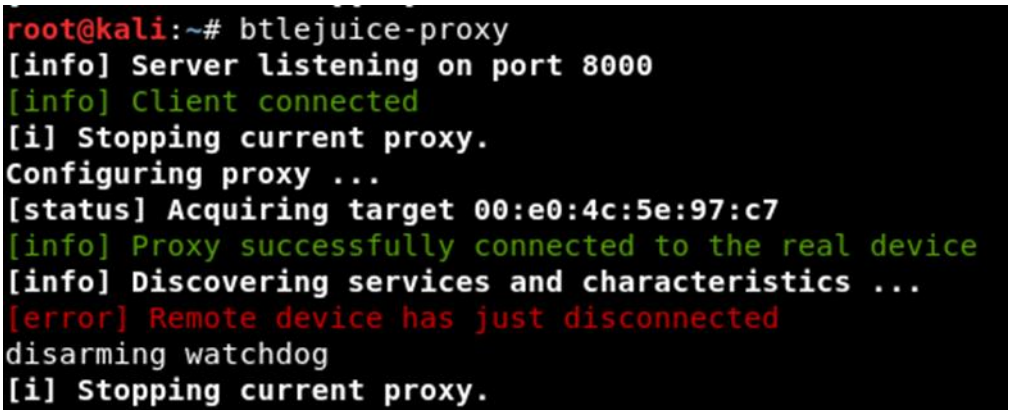


Abbildung 38: BL-5 Btlejuice

Ein Angriff über Gattacker ist ebenso nicht möglich, da das Service File nicht erstellt werden kann um erfolgreich das BDADDR spoofing durchzuführen.

6. Konklusion

Mit den hier verwendeten Methoden und Hilfsmittel war es bei den ausgewählten IoT Geräten nicht möglich eine effiziente Attacke durchzuführen. In allen Fällen mit Ausnahme der BL-5 konnte ein DoS Angriff ausgeführt werden was unter anderem daran liegt, dass die meisten BLE Geräte nur eine aktive Verbindung zulassen. Das Mitlesen des Traffic über Ubertooth konnte zwar in einigen Fällen durchgeführt werden, die Daten waren aber im Versuch nicht nutzbar, da zu keinem der Geräte ein erfolgreicher Schreibbefehl gesendet werden konnte. Um die MitM effizienter zu gestalten ist es möglich mit eigenen Skripten und Anpassungen im BLENO Tool ein BLE Gerät zu klonen. Dies wäre der logische nächste Schritt um die Schwachstellen in den Geräten zu testen.

Allgemein ist zu erkennen, dass auch IoT Geräte für die Sicherheit trivial ist (LED Lampe), ein Mindestmaß an BLE Sicherheit bieten und ohne spezielle Hardware, Software und grundlegendem Wissen über Bluetooth nicht angreifbar sind.

7. Literaturverzeichnis

IEEE Paper

[01] P. Bhagwat: Bluetooth: technology for short-range wireless apps, May/Jun 2001

[02] K. Haataja et al., Bluetooth Security Attacks, 2013

[11] Harry O'Sullivan, Ming Chow. Security Vulnerabilities of Bluetooth Low Energy Technology (BLE), 2015

Bücher

[03] Sashank Pandey, Hacking Internet of Things Bluetooth Low Energy, 2018

Internet

[04] https://github.com/greatscottgadgets/ubertooth/wiki/Build-Guide, 25.05.2018

[05] http://ubertooth.sourceforge.net/hardware/zero/ , 27.05.2018

[06] https://www.heise.de/ct/ausgabe/2017-18-Ubertooth-One-3800397.html, 27.05.2018

[07] https://github.com/DigitalSecurity/btlejuice, 01.06.2018

[08] https://github.com/securing/gattacker, 01.06.2018

[09] https://gattack.io/, 08.06.2018

[10] https://eewiki.net/display/Wireless/A+Basic+Introduction+to+BLE+Security, 17.06.2018

8. Abbildungsquellen

Abbildung 1:

Pravin Bhagwat, Bluetooth: Technology for short range wireless apps, IEEE Internet Computing, May-June 2001, p97

Abbildung 2:

Pravin Bhagwat, Bluetooth: Technology for short range wireless apps, IEEE Internet Computing, May-June 2001, p99

Abbildung 4:

Haataja, K; Hyppönen, K; Pasanen, S; Toivanen, P. Bluetooth Security Attacks, 2013, p6

Abbildung 5:

Haataja, K; Hyppönen, K; Pasanen, S; Toivanen, P. Bluetooth Security Attacks, 2013, p7

Abbildung 6 und 7: Selbsterstellte Tabellen

Abbildung 8 und ff.: Aufgenommene Ergebnisse, während der Selbstversuche