Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

InIT Institut für angewandte
Informationstechnologie

# Projektarbeit Informatik

## BlueBorne War Driving

| | |
|---|---|
| **Autoren** | Benjamin Gehring<br>Béla Horváth |
| **Hauptbetreuung** | Prof. Dr. Bernhard Tellenbach |
| **Nebenbetreuung** | Thomas Sutter |

| | |
|---|---|
| **Datum** | 20.12.2019 |

# Inhaltsverzeichnis

# 1 Formular

Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

## Erklärung betreffend das selbständige Verfassen einer Projektarbeit an der School of Engineering

Mit der Abgabe dieser Projektarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Projektarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

| Ort, Datum: | Unterschriften: |
| --- | --- |
| …… | …… |
| | …… |
| | …… |

Das Original dieses Formulars ist bei der ZHAW-Version aller abgegebenen Projektarbeiten zu Beginn der Dokumentation nach dem Titelblatt mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.

# 2 Zusammenfassung

# 3 Abstract

# 4 Vorwort

Hier folgt eine Danksagung sowie eine Erwähnung aller beteiligten Personen welche zum Erfolg der Arbeit geführt haben.

# 5 Einleitung

## 5.1 Ausgangslage

Das Thema „BlueBorne“ ist keinesfalls neu. Im Jahr 2017 wurden gravierende Sicherheitslücken in der Implementation des Bluetooth Stacks für verschiedenste IoT und Android Geräte entdeckt. Trotzdem finden sich noch immer diverse IoT Geräte, welche genau durch diesen Angriff infiltriert werden können. „BlueBorne“ setzt sich aus den Wörtern „Bluetooth“ und „Airborne“ zusammen und beschreibt einen möglichen, sehr infektiösen Angriff über diverse Schichten des Bluetooth Protokolls. Die gefundenen Lücken ermöglichten das Ausführen von remote Code auf Android und IoT Geräten ohne dass die Geräte miteinander gekoppelt werden mussten. Das Patchen solcher gravierenden Sicherheitslücken ist meist schwierig und dauert sehr lange, da sehr viele IoT Geräte nicht immer auf dem neusten Software Stand sind. Zudem kann die Hardware nicht immer gepachted werden, was die Situation verschlimmert. Die zugrundeliegenden Whitepaper von Armis [6, 5] werden als Einstiegspunkt in die Materie gewählt. Um die technischen Auswirkungen zu sehen, werden die von Armis Labs zur Verfügung gestellten Skripte verwendet. Diese dienen nur zur Veranschaulichung und sind teils auf bestimmte Geräte zugeschnitten. Der aktuelle Stand der Technik zeigt, dass nur eine sehr begrenze Anzahl an Blueborne Scannern existiert. Alle diese Scanner haben die Gemeinsamkeit, dass sie die Geräte nur nach der MAC-Adresse und nicht nach der installierten Softwareversion überprüfen. Dies ist zwar weit einfacher, jedoch liefert es auch viel weniger Informationen über das zu testende Gerät. Der Fokus wird dabei auf den Betriebssystemen Android und Linux liegen. Diese Systeme sind am leichtesten in eine unsichere Version zu bringen wodurch sie sich als Testobjekte eignen. Ausserdem sind diese beiden Systeme die mit am weitesten verbreiteten Systeme im IoT Bereich.

## 5.2 Zielsetzung

Das Ziel dieser Arbeit ist es, über Bluetooth verwundbare Geräte zu finden, ohne diese dabei negativ zu beeinflussen. Dies wird mit ausführlichem passivem Mithören und ggf. aktivem Ansprechen und Testen der Geräte erhalten.

## 5.3 Einschränkung des Scopes

Da der Bluetooth Stack riesig ist und nicht alles behandelt werden kann, wird in dieser Arbeit der Scope auf die Angriffs-Vektoren „Blueborne“ limitiert.

### 5.3.1 Scope: Blueborne

Um das Ziel im definierten Scope zu erreichen, werden folgende Probleme vertieft bearbeitet.

1. Analyse der BlueBorne Sicherheitslücken
2. Analyse von Fingerprinting sowie Scanning Möglichkeiten
3. Konzept für das Fingerprinting und Scanning basierend auf der Analyse
4. Scanning App implementieren
5. Scannen nach BlueBorne verwundbaren Geräten an der ZAHW

## 5.4 Aufgabenstellung

Die Definierte Aufgabenstellung finden Sie im Kapitel 15.1.

## 5.5 Anforderungen

Geräte können aktiv und passiv gescannt werden. Der Scanner wird in Python implementiert.

# 6 Theoretische Grundlagen

## 6.1 BlueBorne

BlueBorne ist der Überbegriff von insgesamt acht Sicherheitslücken in der Bluetooth Implementation von Android, IOS, Linux und Windows Geräten. Alle Geräte welche Bluetooth aktiviert haben sind potentiell davon betroffen. Die Sicherheitslücken wurde von der Firma Armis entdeckt und im September 2017 öffentlich präsentiert. Die Sicherheitslücken konnten auf Linux und Android Geräten zu funktionierenden Exploits umgesetzt werden[1].

## 6.2 Ubertooth

Da viele Bluetooth Geräte im öffentlichen Raum als „unsichtbar“ konfiguriert werden, kann man sie nicht mit Hilfe eines normalen Bluetooth-Adapters erkennen. Um solche Geräte entdecken und auch testen zu können, wird extra Hardware, wie z.B. einen Ubertooth Adapter, benötigt. Mit dem Ubertooth ist es möglich genau diese Geräte im Netzwerk aufzudecken und ihre MAC-Adresse zu bestimmen.

## 6.3 Linux Kernel RCE vulnerability - CVE-2017-1000251

Um diese Sicherheitslücke auszunutzen benötigt man zwei Bluetooth Geräte. Das Gerät welches angegriffen wird muss für diesen Exploit in den „pending“ Modus versetzt werden. Dies erreicht man durch das folglich, vereinfacht beschriebene, Verfahren:

Angreifer sendet und empfängt:

1. Start: Der Angreifer stellt eine Verbindung zum Opfer über den L2CAP-Socket her. Dies muss geschehen um Daten zu schicken und um die DCID des Opfers zu erhalten.
2. Sendet: Configuration Request (DCID = 0x0040 / Destination Channel ID of L2CAP)
3. Sendet: Configuration Request mit dem EFS Element mit stype = L2CAP_SERV_NOTRAFIC
4. Empfängt: Configuration Request von Opfer mit DCID = 0x0040

[1] Alle Demonstrationen sind Proof of Concept implementationen.

5. Sendet: Configuration Response mit Result Feld = L2CAP_CONF_PENDING und den wiederholten Parametern (im Beispiel die MTU) um den Stackoverflow zu generieren.

Bei dieser Attacke ist es sehr wichtig das alle gesendeten Pakete welche den Stackoverflow generieren sollen L2CAP konforme Pakete darstellen.

## 6.4 BlueZ information leak vulnerability- CVE-2017-1000250

Die BlueZ information leak Sicherheitslücke beschreibt einen Fehler in der Implementation des BlueZ Servers auf Linux Basis. Der Fehler liegt hierbei in der Abhandlung von fragmentierten Paketen welche nicht als ganzes übermittelt werden können. Zu finden ist er in der Funktion: “service_search_attr_req„.

```
typedef struct {
        uint32_t timestamp;
        union {
                uint16_t maxBytesSent;
                uint16_t lastIndexSent;
        } cStateValue;
} sdp_cont_state_t;
```

Abbildung 1: cState-struct welcher maxBytesSent enthält. Wird vom Angreifer kontrolliert.

```
} else {
        /* continuation State exists -> get from cache */
        sdp_buf_t *pCache = sdp_get_cached_rsp(cstate);
        if (pCache) {
                uint16_t sent = MIN(max, pCache->data_size - cstate->cStateValue.maxBytesSent);
                pResponse = pCache->data;
                memcpy(buf->data, pResponse + cstate->cStateValue.maxBytesSent, sent);
                buf->data_size += sent;
                cstate->cStateValue.maxBytesSent += sent;
                if (cstate->cStateValue.maxBytesSent == pCache->data_size)
                        cstate_size = sdp_set_cstate_pdu(buf, NULL);
                else
                        cstate_size = sdp_set_cstate_pdu(buf, cstate);
        } else {
                status = SDP_INVALID_CSTATE;
                SDPDBG("Non-null continuation state, but null cache buffer");
        }
```

Abbildung 2: Betroffene Methode service_search_attr_req in der SDP-Server implementation.

Der in Abbildung 1 gezeigte „Continuation State“ sollte eigendlich nur vom Server verwaltet werden können, wird jedoch jedes mal an den Client mitgeschickt. Der Server geht hier davon aus, dass der Client den „Continuation State“ unverändert, an seine Response gepackt, zurücksendet. Verändert man jedoch die Werte „maxBytesSent“ und „lastIntexSent“ so kann man den Server dazu bringen aus einem falschen Index (von einer falschen Adresse im Speicher) zu lesen. Um dies erfolgreich durchzuführen muss vom Client nur die „If-Abfrage“, gezeigt in Abbildung 2 umgehen, in dem er „maxBytesSent“ anpasst.

## 6.5 Android information Leak vulnerability - CVE-2017-0785

Diese Sicherheitslücke wurde im SDP Protokoll des Bluetooth Stacks gefunden und ermöglicht es, an bestimmte Speicherdaten eines Gerätes zu gelangen. Das SDP Protokoll teilt anderen Geräten mit, welche Bluetooth Services das angefragte Gerät unterstützt. Dabei stellt das externe Gerät (Client) einen Request der vom eigenen Gerät (Server) mit einer Response beantwortet wird. Diese Response kann maximal die Grösse der MTU des Clients annehmen und muss ansonsten fragmentiert werden. Der Fehler liegt nun im Abhandeln dieser Responses, wobei der Server berechnet welche Fragmente noch gesendet werden müssen. Der Server schickt im Falle einer Fragmentierung einen Continuation State mit der Response zu dem Client. Der Client sendet darauf einen identischen Request nur mit diesem Continuation State als Zusatz. Der Server weiss nun welches Fragment er als nächstes schicken muss. Ist z.B. die MTU 50 Bytes gross und die Response 80 Bytes sind 2 Fragmente nötig. Die Variable „remaining_handles“ berechnet sich aus „number_response_handles“ und „continuation_offset“.

```
rem_handles =
    num_rsp_handles - cont_offset; /* extract the remaining handles */
```

Abbildung 3: Berechnung der rem_handels Variable

Schafft man es nun, dass number_response_handels kleiner ist als der continuation_offset erreicht man einen Underflow. Um dies zu erreichen benötigt man zwei Verbindungen man geht nun wie folgt vor :

1. Im ersten Schritt stellt man eine Verbindung zu einem beliebigen Service her, welcher eine Response schickt die grösser als die angegebene MTU ist.

2. Den dabei erhaltenen Continuation State verwendet man für den zweiten Schritt, in welchem eine Verbindung zu einem beliebigen Service aufgebaut wird der eine kleinere Response als die MTU zurückschickt.

3. Innerhalb der Berechnung der „remaining_handels“ Variable wird nun z.B. 2 - 1 gerechnet. Dies führt bei einem uint16 Feldtyp zu einem Underflow wodurch der Server eine sehr grosse Anzahl an Responses zurückschickt.

4. All diese Responses werden mit einer Schleife abgefangen und enthalten wichtige Informationen des Gerätespeichers, welche man normalerweise nicht einsehen kann.

Dieser Leak kann in einem weiteren Vorgehen dazu genutzt werden den ASLR Schutzmechanismus gegen BufferOverflows zu umgehen. Dazu wird ausgehend von einer geleakten Speicheradresse ein Offset zurück auf die Basisadresse der Bluetooth Bibliotheken berechnet. Während des Angriffs wird ebenfalls diese Sicherheitslücke genutzt um mithilfe des Offsets die Basisadresse zu berechnen.

## 6.6 Android RCE vulnerability Nr. 1 - CVE-2017-0781

Die Schwachstelle auf Android Geräten liegt im BNEP Protokoll von Bluetooth. Hierbei sei angemerkt, dass Android auf dem Bluedroid Stack basiert und nicht auf dem BlueZ Stack von Linux. Diese verwenden keine gemeinsamen Codeteile. Das BNEP Protokoll wird genutzt um IP Pakete zwischen zwei Geräten auszutauschen. Ein Anwendungsfall wäre z.B. die Nutzung eines gemeinsamen Mobilfunkverbindungshotspots über Bluetooth. BNEP fügt dazu einen Header vor der Ethernet Payload an. Zusätzlich werden auch Mechanismen, wie Flussteuerung über sogennante Control Messages ermöglicht. Solche Nachrichten sind durch ein Extension Header beschrieben4, welcher wiederum mit einem Extension Bit angekündigt wird.

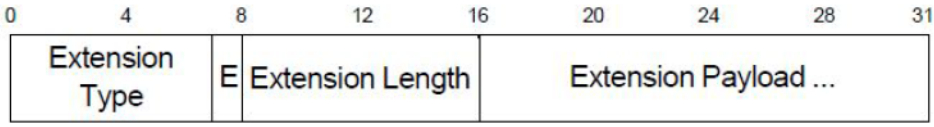


Abbildung 4: Der Extension Header des BNEP Protokolls

In der Implementation der Abhandlung solcher Control Message wurde nun ein entscheidender Fehler gefunden, der ein BlueBorne Angriff möglich macht.

Im abgebildeten Codeabschnitt 5 wird eine eingehende BNEP Control Message entgegengenommen und verarbeitet. Dabei wird innerhalb des if-Statments die unverarbeitete Nachricht für die spätere Nutzung in *p_pendig_data* zwischengespeichert. Innerhalb dieses Vorgangs (memcpy) verbirgt sich allerdings ein entscheidender Fehler: Die unverarbeitete Nachricht wird an die Stelle *p_pending_data+1* kopiert was natürlich zu einem Overflow führt da die Speicheradresse + 1 nach *p_pendig_data* nicht alloziert wurde. Der Overflow entspricht der Grösse rem_len. Um in diesen Codeabschnitt zu gelangen kann folgende Payload6 als BNEP Nachricht gesendet werden:

```
UINT8 *p = (UINT8 *)(p_buf + 1) + p_buf->offset;
...
type = *p++;
extension_present = type >> 7;
type &= 0x7f;
...
switch (type)
{
...
case BNEP_FRAME_CONTROL:
    ctrl_type = *p;
    p = bnep_process_control_packet (p_bcb, p, &rem_len, FALSE);
    if (ctrl_type == BNEP_SETUP_CONNECTION_REQUEST_MSG &&
        p_bcb->con_state != BNEP_STATE_CONNECTED &&
        extension_present && p && rem_len)
    {
        p_bcb->p_pending_data = (BT_HDR *)osi_malloc(rem_len);
        memcpy((UINT8 *)(p_bcb->p_pending_data + 1), p, rem_len);
        ...
    }
...
```

Abbildung 5: Der fehlerhafte Codeabschnitt im BNEP Protokoll

| type | ctrl_type | len | Overflow payload (8 bytes) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 81 | 01 | 00 | 41 | 41 | 41 | 41 | 41 | 41 | 41 | 41 |

Abbildung 6: Die benötigte Payload um den BufferOverflow auszulösen

Der „Type“ Wert 81 entspricht dem Extension Bit für die Control Message und 01 unter ctrl_type setzt den Control Type der Message auf BNEP_FRAME_CONTROL, welcher uns ermöglich in den korrekten Switch Case zu gelangen.

Für einen erfolgreichen Angriff auf ein Android Gerät müssen noch folgende zusätzliche Schritte befolgt werden:

1. Umgehung des ASLR Schutzmechanismus von Android gegen BufferOverflows
2. Shell Code im Memory des Gerätes platzieren der durch den Overflow ausgeführt werden kann
3. Entsprechenden Codeabschnitt überschreiben der diesen Shell Code ausführt

# 7 Analyse der BlueBorne Sicherheitslücken

Insgesamt umfasst BlueBorne 8 Sicherheitslücken. In dieser Arbeit wurden diese auf die Systeme Android und Linux beschränkt. Grund dafür ist die grössere Informationsverfügbarkeit und der Umstand dass diese Systeme bei IoT Geräten stark verbreitet sind.
Es handelt sich sowohl auf Linux als auch auf Android um einen Programmierfehler, welcher zu einem BufferOverflow führt. Dieser kann wiederum genutzt werden um Schadcode im System zu platzieren.

## 7.1 Angriff auf Android

### 7.1.1 Informationsbeschaffung

Um einen generellen Überblick über das Thema zu erhalten, wurden in einem ersten Schritt die offiziellen WhitePaper [6] der Entdecker von BlueBorne studiert. Ausserdem wird auf Beispielskripte verwiesen die einen Angriff auf ein Nexus 5X ermöglichen. Aus Gründen der einfachen Reproduzierbarkeit, wurde für diese Projektarbeit dasselbe Gerät als Testgerät beschafft. Ziel in einem nächsten Schritt, sollte es nun sein, diesen Angriff nachzubilden.

Der Beispielangriff funktioniert gemäss Beschreibung allerdings nur für genau diese Android Version (v7.1.2) und dieses Gerät. Bei abweichenden Spezifikationen müssen leichte Anpassungen vorgenommen werden, die in den WhitePaper allerdings nicht ausreichend beschrieben sind.
Aus diesem Grund wurde weitere Informationsbeschaffung durch Keywordsuche auf Google betrieben. Dies führte zu einem Blogpost [2] der die Vorgehensweise für denselben Angriff auf Version 6.0.1 beschreibt. Allerdings wird dabei ein anderes Testgerät verwendet. Nach erster Analyse des Artikels wurde allerdings kein Grund gefunden, warum die beschriebenen Schritte, nicht auch auf dem Nexus 5X umgesetzt werden könne. Aus diesem Grund und weil das komplette Vorgehen mitsamt erfolgreichem Angriff beschrieben wird, entschieden wir uns diesen Artikel zu verwenden um einen Angriff auf Android Version 6.0.1 zu starten.
Da die Scanner Applikation für möglichst viele Versionen und Geräte funktionieren soll war dieser Schritt unserer Meinung nach nötig.

### 7.1.2 Geräteauswahl

Die Geräteauswahl fiel uns einfach. Die gewählten Unterlagen bezogen sich auf ein Nexus 5X und dieses Gerät war leicht zu beschaffen. Aus diesen Gründen wurde dieses Gerät gewählt.

### 7.1.3 Methoden und Vorgehen

ArmisLab hat auf GitHub ein Verzeichnis mit Code veröffentlicht[2]. Dieses Repository umfasst Python Skripte, mithilfe deren, der Angriff für bestimmte Geräte direkt durchführbar wird. Für den Android Exploit ist ein Nexus 5X mit Android 7.1.2 nötig. Auf dem Nexus 5X wurde die Android Version 7.1.2 installiert.
Für den Angriff auf Version 6.0.1 mussten im Skript vier Variablen angepasst werden, welche Adressen des Zielgeräts darstellen, zu sehen in Abbildung 7.

```
#For Nexus 5X 7.1.2 patch level Aug/July 2017
LIBC_TEXT_STSTEM_OFFSET = 0x45f80 + 1
LIBC_SOME_BLX_OFFSET = 0x1a420 + 1

# Aligned to 4 inside the name on the bss (same for both supported phones)
BSS_ACL_REMOTE_NAME_OFFSET = 0x202ee4
BLUETOOTH_BSS_SOME_VAR_OFFSET = 0x14b244
```

Abbildung 7: Die Benötigten Adressen Offsets für den Blueborne Angriff

Diese Adressen dienen dazu, den ASLR Schutzmechanismus zu umgehen, welcher eine erfolgreiche Nutzung des BufferOverflows verhinderte. Die benötigten Werten werden aus dem Speicher es Telefons ausgelesen. Der Zugriff auf den Speicher wird durch das anfügen eines Debuggers[3] ermöglicht [7]. Daraus konnten anschliessend Offsets berechnet werden. Diese ermöglichen es, mithilfe des Information Leaks beschrieben im Kapitel 6.5, auf die Basiadresse der Bibliotheken *Libc.so* und *bluetooth.default.so* zurück zu rechnen. Um die Werte auszulesen, ist ein Gerät in gleicher Version und vom gleichen Herstellern notwendig, da sich die Adressen in anderen Versionen beziehungsweise Herstellern unterscheiden. Ausserdem waren Root Privilegien auf dem zu testenden Gerät notwendig. Wir installierten also die Version 6.0.1 mit Root Berechtigungen und gingen gemäss den Anweisungen im Blogpost vor.

### 7.1.4 Resultate

Zu Beginn hatten wir Probleme das Beispielskript mit dem Exploit für die Android Version 7.1.2 auszuführen. Dies lag daran dass innerhalb des Skriptes die BDADDR des eigenen Bluetooth Adapters geändert wurde. Standardmässig eingebaute Adapter in den Laptops unterstützten eine solche Funktion nicht. Wir mussten uns also einen entsprechenden CSR Bluetooth Adapter besorgen, welcher diese Möglichkeit bot.
Mit dem neuen Adapter gelang es uns anschliessend sehr schnell den Angriff selbst durchzuführen und wir erhielten den kompletten Zugriff auf unser Testgerät.

[2]Github Repository: https://github.com/ArmisSecurity/blueborne
[3]PEDA-Arm: https://github.com/alset0326/peda-arm

Bei der Version 6.0.1 bot der verwendete Blogpost [2] eine solide Ausgangslage, jedoch wurde nicht alles vollständig beschrieben. Es gelang uns, mit dem beschriebenen Vorgehen, drei der vier benötigten Variablen zu berechnen.
Es scheiterte jedoch an der Variable „BSS_REMOTE_NAME_OFFSET“. Dieser Offset sollte über die Position des Namens des gekoppelten Bluetooth Geräts im Speicher berechnet werden. Diese Position muss mittels der Funktion „searchmem“ ermittelt werden, welche auch nach mehreren Versuchen keine Ergebnisse lieferte.

Da ohne diese Informationen die Offsets nicht berechnet werden konnten und auch keine andere Möglichkeit gefunden wurde, war ein erfolgreicher Angriff auf v6.0.1 nicht möglich. Aus diesem Grund wurde als nächsten Schritt versucht die Prozedur auf v7.1.1 zu wiederholen um eventuelle Probleme mit Version 6.0.1 auszuschliessen. Auf der Version 7.1.1 gelang es uns den BufferOverflow durchzuführen und die Suchfunktion „searchmem“ gab uns ebenfalls entsprechende Ergebnisse zurück. Wir konnten erfolgreich den Schadcode im Speicher platzieren, allerdings gelang es uns nicht diesen Auszuführen. Die Vermutung liegt nahe, dass die Systemfunktion, welche auf Android 7.1.2 dafür verwendet wurde, eine andere war als in der Android Version 7.1.1.

### 7.1.5 Diskussion

Weiterführend kann man sicherlich eine alternative Systemfunktion suchen, welche die Ausführung ermöglicht (wie es bereits im Blogpost für v6.0.1 getan wurde). Wir genügten uns an diesem Punkt allerdings mit dem Fortschritt, da wir der Meinung waren, dass ein erfolgreicher BufferOverflow ausreicht um eine Scanner Applikation entwickeln zu können und für diesen wurden die Adressen nicht benötigt. Wir konnten diesen auch auf allen anderen Versionen von Android (v6.0.1 einschliessend solange tiefer oder gleich v7.1.2) ausführen und hatten einen eindeutigen Hinweis auf die Verwundbarkeit des Gerätes. Die eigentliche Übernahme des Gerätes, war für das Ziel der Arbeit somit gar nicht nötig.

## 7.2 Angriff auf Linux

### 7.2.1 Informationsbeschaffung

Es wurden, gleich wie beim Angriff auf Android, die Whitepaper [5] von ArmisLab als Grundlage gewählt. Aus einer Google Keywordsuche[4] wurden keine weiteren Artikel gefunden. Einzig alternative Skripts und Implementationen des Angriffes konnten studiert und getestet werden.

[4] Es wurde nach CVE-2017-1000251 seucht, um alternative implementationen des Exploits zu finden.

### 7.2.2 Geräteauswahl

Die Geräteauswahl unter Linux war nicht gleich einfach. Wir einigten uns auf einen Raspberry Pi 3b+ da dieser Bluetooth fähig, leicht auf eine unsichere Software Version zu bringen, und nicht schwer zu beschaffen ist. Die Unterlagen, auf welche wir uns stützten, verwendeten eine Smartwatch und ein Amazon Echo, welche beide jedoch schwer zu beschaffen und schwer auf eine unsichere Version zu setzen sind.

### 7.2.3 Methoden und Vorgehen

Auf dem Raspberry Pi 3b+ wurde eine veraltete Ubuntu Mate Version 16.04 installiert. Diese Version verwendet einen Linux Kernel v. 4.1.19-v7+ und Bluethooth(BlueZ) v. 5.37. Beides davon ist veraltet und somit noch von den beschriebenen Sicherheitslücken im Kapitel 6.3 und 6.4 betroffen. Um das Prinzip des Exploits nachzuvollziehen wird nur die Bufferoverflow Sicherheitslücke intensiver getestet. Sobald der Bufferoverflow ohne Stack Canaries durchgeführt werden kann, kann auch der Vollzugriff auf das Gerät in weiteren Schritten erfolgen. Diese weiteren Schritte werden für das Verständnis zwar analysiert, jedoch nicht getestet.

### 7.2.4 Resultate

### 7.2.5 Diskussion

# 8 Analyse Fingerprinting und Scanning Methoden

Neben der Nutzung der BlueBorn Sicherheitslücken selbst, gibt es weitere Ansätze, die zur Erkennung von möglicherweise verwundbaren Geräten genutzt werden können. Diese zielen beispielsweise auf die Erkennung der verwendeten Bluetooth-, Betriebssystem-, oder auch Bluetoothstack-Version ab. Um die Version zu erkennen nutzt man Verhalten aus, welches spezifisch für eine bestimmte Version ist und welches mittels passiver (zum Beispiel Beobachten des Netzwerkverkehrs eines Geräts) oder aktiver Methoden (direkte Interaktion mit dem Gerät) festgestellt werden kann. Dieses messbare spezifische Verhalten stellt also quasi ein Fingerabdruck für eine Version dar. Dieses Vorgehen fasst man deshalb auch unter dem Begriff „Fingerprinting“ (von Geräten, Software-Versionen und weiteres) zusammen. Nachfolgend analysieren wir, welche im Hinblick auf das Ziel „verwundbare Geräte finden“ Fingerprinting Ansätze es bereits gibt. Dazu führen wir eine Literaturrecherche durch, deren Methodik und Ergebnisse wir in den nächsten zwei Abschnitten präsentieren. Im letzten Abschnitt ordnen wir das Ergebnis ein, insbesondere auch bezüglich möglicher Einschränkungen bezüglich dessen Gültigkeit und Vollständigkeit.

## 8.1 Methoden und Vorgehen

### 8.1.1 Literaturrecherche Online

Die Literaturrecherche wurde mittels Google und Schlagwörtern durchgeführt. Die Suche wurde nur an Geräten welche sich im ZHAW-Netzwerk befinden durchgeführt, um Arbeiten zu finden, für welche man einen Bildungsnetzwerk-Zugang benötigt.

| **Schlagwort** | **Gefundene Artikel** | **Relevanz** |
|---|---|---|
| Bluetooth Fingerprinting | Remote Device Identification based on Bluetooth Fingerprinting Techniques[1] | SDPTool wird für Fingerprinting getestet. |
| Bluetooth OS Fingerprinting | bluetooth device security database[5] | Veraltete Datenbank auf SDPTool Basis, unbrauchbar in diesem Zustand. |
| OS Fingerprinting | Studying Bluetooth Malware Propagation: The BlueBag Project[6] | Zeigt verschiedenste Angriffe auf, ist nicht weiter interessant für das Ziel. |
| Ubertooth fingerprint device | An overview of bluetooth device discovery and fingerprinting techniques – assessing the local context[7] | Verlgleicht discovery/fingerprinting Methoden, jedoch wurden die wichtigen schon anderweitig entdeckt. |

### 8.1.2 Bücher und Literatur

Nebst der online Literatur wurden auch allgemeinere Informationen zu Bluetooth aus gedruckter Fachliteratur gezogen.

| **Buch** | **Gefundene Information** | **Relevanz** |
|---|---|---|
| Hacking und Security - Das umfassende Handbuch[3] | Bluetooth | Allgemeine Informationen sowie vorhandene Tools für Scans entdeckt. |

[5] Quelle: http://www.betaversion.net/btdsd/, Zugang: 11/2019
[6] Quelle: https://ieeexplore.ieee.org/abstract/document/4140986, Zugang: 11/2019
[7] Quelle: https://pdfs.semanticscholar.org/3c3d/a39c67e387bca95b2fe6752777ce10373656.pdf, Zugang: 11/2019

## 8.2 Auswertung der Recherche

Folgend wird kurz besprochen, welche Teile der gefundenen Literatur weiter verwendet werden. Weiter verwendet werden nur Informationen, die bezüglich dem Ziel „Fingerprinting über Bluetooth“ einen neuen Ansatz bieten.

### 8.2.1 Online Literaturquellen

Die online Literaturrecherche viel etwas ernüchternd aus. Trotz vieler gefundener Paper und Arbeiten waren die verfolgten Fingerprinting-Methoden oft dieselben oder nicht für das gesetzte Ziel dieser Arbeit anwendbar. Ein sehr spannender Ansatz wurde im Artikel „Remote Device Identification based on Bluetooth Fingerprinting Techniques“[1] besprochen, wobei eine lokale Fingerprinting Datenbank geführt wurde. Neue Geräte werden mittels dem SDPTool angefragt und deren Fingerprint als Hash von bestimmten Zeilen der SDPTool-Ausgabe dargestellt. Dieser Ansatz könnte vielversprechend sein, jedoch ist die Datenbank aus dem Jahr 2004 und somit schon längst nicht mehr aktuell.

### 8.2.2 Bücher

In der verwendeten Literatur wurden diverse Bluetooth monitoring und penetration testing Tools vorgestellt, sowie deren Stärken und Schwächen besprochen. Genauer analysiert wurden die Tools „Blue_hydra, blueranger, hcitool, sdptool, hcidump, ubertooth“. Welche und wie diese Tools implementiert wurden wird in den nachfolgenden Kapiteln beschrieben

# 9 Konzept der Fingerprinting und Scanning Methoden

Mit den Ausgewählten Fingerprinting Methoden wurden das erste Konzept definiert. Das Ziel diese ist, passiv mitzuhören ohne die Geräte negativ zu beeinflussen. Ein weiteres Ziel ist natürlich möglichst viele Informationen über die entdeckten Geräte zu erhalten welche diese klassifizieren könnten. In diesem Kapitel werden aus der Analyse ausgewählte Methoden detaillierter beschrieben und Begründungen für die Verwendung geliefert.

## 9.1 Methoden und Vorgehen

In dieser Sektion werden verschiedene versuchte Ansätze der Klassifikation beschrieben. Es wird auf Tools verwiesen und es werden Informationen geliefert, wieso gewisse Ansätze verfolgt wurden und andere nicht.

### 9.1.1 Unsichtbare Geräte - Verbindungsaufbau

Dank dem Ubertooth One können Unsichtbare Geräte entdeckt werden. Der Vorgang wird besser in der Abbildung 8 dargestellt. Wir beziehen uns hierbei auf einen Blogpost von Michael Ossmann [4] einem der Entwickler von Ubertooth und anderen Gadgets.

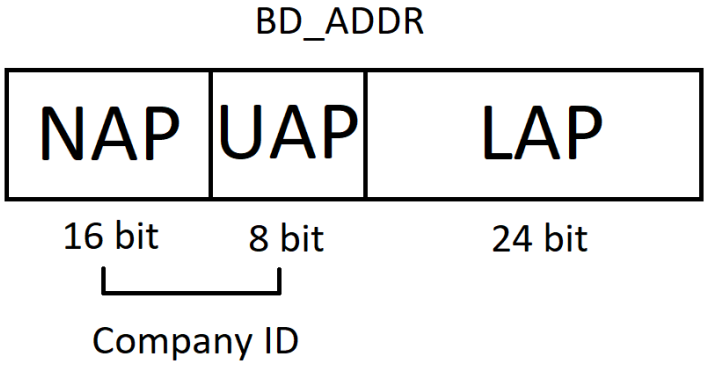


Abbildung 8: Der Aufbau einer Bluetooth Adresse mit eigenen Bezeichnungen

Hierbei ist wichtig zu verstehen, dass nur der LAP Teil der MAC-Adresse („Lower Address Part“), durch aktives auslesen der übermittelten Bluetooth Packete in der Umgebung, vom Ubertooth erkannt wird. Der UAP, der „Upper Address Part“ kann vom Ubertooth, durch passives mithören, errechnet werde. Hat man den UAP errechnet, so reicht diese fast komplette MAC-Adresse in der Form von „00:00:XX:XX:XX:XX:XX“

für die meisten Dienste aus und es kann mit dem Gerät kommuniziert werden. Um jedoch ein Gerät genau einem Hersteller zuordnen zu können, benötigt man auch den richtigen NAP, auch „Non-significant Address Part“ genannt. Hierfür wird eine MAC-Adressen Lookup Tabelle [8] verwendet. Die Idee besteht darin, den UAP Teil der zu findenden Adresse mit dem Ubertooth zu errechnen. Nachdem dieser erfolgreich errechnet wurde, könnte Bruteforce angewendet werden, um alle möglichen MAC-Adressen aus der Lookup Tabelle durchzutesten, welche das Format „XX:XX:UAP:LAP:LAP:LAP“ besitzen[9].
Dieses Vorgehen kann jedoch sehr viel Zeit in Anspruch nehmen, woduch versucht wird, über den Gerätenamen (der bei den meisten Geräten standarmässig den Herstellernamen beinhaltet) die Anzahl möglicher MAC-Adressen einzuschränken. Hier wird jedoch die Annahme getroffen, dass sich das anzugreifende Gerät auch in einem Modus befindet, in welchem es auf Verbindungsanfragen antwortet. Ist dies nicht der Fall, so kann die Geräte MAC-Adresse nicht genauer bestimmt werden.

### 9.1.2 Link Manager Protocol Version

Über das Linux eigene „hcitool“[10] kann man Bluetooth Geräte konfigurieren und mit anderen Geräten kommunizieren. Dieses Tool bietet unter anderem eine „info“ Funktion über welche man gerätespezifische Informationen abrufen kann. Über diese Infopage erhält man auch auskunft über die verwendete LMP Version, von welcher aus man Rückschlüsse auf die Bluetooth Version ziehen kann.

```
LMP Version: 4.0 (0x6) LMP Subversion: 0x132
```

Abbildung 9: Output von „hcitool info“ für LMP Version

Der Interessante Teil der LMP Version liegt hierbei im Ausdruck in der Klammer. (0x8) würde hierbei zur LMP Version 8 gemapped werden, somit werden vom Gerät alle Bluetooth Versionen von 4.2 und absteigend unterstützt.

## 9.2 Resultate

[8] Verwendete MAC-Adressen Tabelle: https://linuxnet.ca/ieee/oui/nmap-mac-prefixes
[9] UAP und LAP sind hier nur Platzhalter für die richtigen MAC-Adressen Teile
[10] Hcitool Dokumentation: https://linux.die.net/man/1/hcitool

| LMP | Bluetooth Version |
|---|---|
| 0 | Bluetooth 1.0b |
| 1 | Bluetooth 1.1 |
| 2 | Bluetooth 1.2 |
| 3 | Bluetooth 2.0 + EDR |
| 4 | Bluetooth 2.1 + EDR |
| 5 | Bluetooth 3.0 + HS |
| 6 | Bluetooth 4.0 |
| 7 | Bluetooth 4.1 |
| 8 | Bluetooth 4.2 |
| 9 | Bluetooth 5 |
| 10 | Bluetooth 5.1 |

Abbildung 10: Mapping Tabelle für LMP Versionen zu Bluetooth Versionen

# 10 Scanner App Implementation

Die Scanner Applikaton ist eigentliches Ziel dieser Arbeit. Sie soll umligende Geräte über Bluetooth erkennen und feststellen ob dessen Software anfälligen für einen BlueBorne Angriff ist.

## 10.1 Methoden und Vorgehen

In einem ersten Schritt wird eine Blueborne Scanner App in Python entwickelt. Diese scannt sichtbare oder unsichtbare Geräte (mit Ubertooth) über Bluetooth. Des weiteren klassifiziert die App die entdeckten Bluetooth Geräte nach deren MAC-Adressen. Weiter wird versucht über „OS-Fingerprinting“ mehr Informationen über das gegenüberliegende System zu erhalten. Falls der Benutzer beschliesst ein Gerät zu prüfen, sendet die Applikation eine Payload, welche einen BufferOverflow im Bluetooth Service verursacht, falls das Gerät verwundbar ist. Die App erkennt einen erfolgreichen BufferOverflow daran, dass der Service für kurze Zeit nicht mehr antwortet. Der Benutzer wird über das Ergebniss informiert und kann gegebenenfalls weitere Massnahmen treffen.

### 10.1.1 Grundlegende Funktionalitäten

Für die grundlegende Funktionalitäten versuchten wir als erstes die Möglichkeit zu implementieren, nach umliegenden, sichtbaren Geräten zu scannen und diese auf Verwundbarkeit zu testen.
Nach dem Scann werden dem Nutzer die Geräte angezeigt, wobei er eines auswählen kann, welches nun genauer untersucht werden soll. Die gefundene Bluetooth Adresse wird dazu verwendet, eine BNEP Verbindung zum Gerät aufzubauen und die schädliche Payload zu senden. Die Applikation soll danach prüfen ob das Gerät noch

Antwortet. Ist dies nicht der Fall, gehen wir davon aus, der BufferOverflow war erfolgreich und der Bluetooth Service ist abgestürzt. Das ist ein eindeutiger Hinweis auf die Verwundbarkeit des Geräts und wird dem Benutzer angezeigt.

### 10.1.2 Weitere Geräteinformationen

### 10.1.3 Unsichtbare Geräte finden

Um Geräte zu finden die zwar Bluetooth eingeschalten haben aber nicht sichtbar sind, wurde in der Scanner Applikation eine Ubertooth kompabilität implementiert. Diese kann mit den Parametern -u -t [TIME] aktiviert werden. Für [TIME] stehen drei Möglichkeiten (-s, -m, -l) zur Verfügung. Die Option -s lässt den Ubertooth nur kurz scannen (20s für LAP und 10s pro Gerät für UAP) und stellt eine schnelle aber ungenaue Abhörmethode dar. Mit -l scannt der Ubertooth lange (40s für LAP und 40s pro Gerät für UAP) was eine langsame aber genaue Möglichkeit bietet für einen Scan. Der Parameter -m (30s LAP und 20s UAP) ist ein Mittelweg zwischen den beiden vorherigen.
In einem ersten Schritt fängt der Ubertooth alle Bluetooth Packete in der Umgebung ab und liest die LAPs aus. Erscheint eine LAP mehrere Male handelt es sich mit grosser Wahrscheinlichkeit um ein kommunzierendes Gerät. Die gefundene LAP wird in einer Liste gespeichert.
Im zweiten Schritt wird versucht pro gefundene LAP bzw. Gerät die UAP zu berechnen. Dafür fokussiert sich der Ubertooth nur auf die Packete mit der entsprechenden LAP. Der Vorgang muss für jedes Gefundene Gerät aus der Liste, wiederholt werden und kann desshalb relativ lange brauchen (Je nach angegebener [TIME]). Leider lässt sich der Ubertooth nicht so konfigurieren, dass er nach gefundener UAP die Berechnung abbricht und mit dem nächsten Gerät fortfährt. Es wird also für jedes Gerät immer für die volle, konfigurierte Zeitspanne gescannt. Durch diese Problematik lässt sich keine pro Gerät unterschiedliche Zeitspanne programmieren. Wie lange es dauert bis die UAP gefunden wird ist sehr verschieden und es kann vorkommen das auch mit dem -l Paramter (40s pro Gerät) keine UAP gefunden wird.
Als dritten Schritt werden dem Benutzer nun alle gefundenen Geräte mit BDADDR (im Format UAP:LAP:LAP:LAP) und Name dargestellt. Geräte dessen UAP nicht gefunden wurde, werden nicht dargestellt, da mit diesen keine Interaktion möglich ist. Der Nutzer kann nun ein Gerät auswählen, für welches nachfolgend die NAP gesucht und das Gerät auf Verwundbarkeit getestet wird.
Die NAP wird in einem zweistufigen BruteForce Ansatz versucht zu ermitteln. Es wird eine Liste mit allen Herstellernspezifischen Adressen, aus einer Datei, eingelesen. Diese Adressen bestehen aus NAP:NAP:UAP und können eindeutig einem Hersteller zugewiesen werden. Die Scanner Applikation sucht also alle Adressen mit derselben UAP in der Liste. Trotz diesen Einschränkungen können es noch immer über 100 zutreffende Adressen sein. Aus diesem Grund wird in einer ersten Stufe versucht der gefundene Gerätename einem Hersteller der Adressen zuzuordnen. Ist diese erfolgreich wird die gefundene, komplette Adresse darauf geprüft ob ein Verbin-

dungsaufbau möglich ist. Trifft dies zu kann das Gerät auf Verwundbarkeit getestet werden. Im anderen Fall oder falls der Gerätename nicht einem Hersteller zugeordnet werden konnte, werden alle gefundenen NAPs mit der enstprechenden UAP und LAP verwendet um einen Verbindungsaufbau zu versuchen. Jene NAP die in Kombination mit der UAP und LAP in einem erfolgreichen Verbindungsaufbau endet ist die gesuchte und kann im letzen Schritt für den Test verwendet werden. Der Vorgang kann pro Adresse ca. 4 Sekunden dauern.
Es sei an dieser Stelle angemerkt, dass die unsichtbaren Geräte sowohl während des LAP Scans und auch während des UAP Scans kommunizieren müssen um vom Ubertooth erkannt zu werden. Ist dies nicht der Fall ist eine Erkennung unmöglich.

## 10.2 Resultate

# 11 Resultate

## 11.1 Analyse Blueborne

## 11.2 Analyse Fingerprinting

## 11.3 Scanner Applikation

## 11.4 Wardriving ZHAW

# 12 Diskussion und Ausblick

## 12.1 Ausblick

1. Datenbank befüllen mit SDPTOOL scans wodurch man eventuell auch Geräte mit verändertem Bluetooth namen klassifizieren könnte. BelaPhone ist ein LG G6 usw. dafür müsste man jedoch die Datenbank selbst erstellen mit testgeräten.

# 13 Verzeichnisse

# Literatur


[1] M. Herfurt and C. Mulliner. Remote device identification based on bluetooth fingerprinting techniques, December 2004.

[2] A. Jesús. Blueborne rce on android 6.0.1 (cve-2017-0781), Juni 2018.

[3] M. Kofler et al. *Hacking und Security*, pages 281–297. Rheinwerk Verlag, Bonn, 2018.

[4] M. Ossmann. Discovering the bluetooth uap, Juni 2014.

[5] B. Seri and A. Livne. Exploiting blueborne in linux based iot devices, 2019.

[6] B. Seri and G. Vishnepolsky. Blueborne, September 2017.

[7] B. Tellenbach. *Exploitation – Assembly and GDB Refresher*. ZHAW/SOE/INIT, 2019.

## 13.1 Glossar

| Begriff | Erklärung |
|---|---|
| ASLR | Address space layout randomization beschreibt einen Sicherheitsmechanismus in Computersystemen, welcher das Ausnutzen von Speicherkorruption verhindern soll. Um einen Angreiffer daran zu hindern auf eine Adresse einer korrupten Systemfunktion zu springen, werden bei jedem Neustart des Gerätes diese Adressen randomisiert. |
| CVE | Eine Liste, gefüllt mit Einträgen für öffentlich bekannte Sicherheitslücken |
| Exploit | Eine Sicherheitslücke (Vulnerabilität) wird auch Exploit genannt. Ein Exploit ist ein nicht absichtlich eingeführter und unbehobener Fehler im Softwarecode, welcher von Angreifern ausgenutzt werden kann, um das System zu infiltrieren oder Malware einzuschleusen. |
| MAC-Adresse | Die MAC-Adresse ist die Eindeutige Hardware Adresse eines Gerätes. Sie ist zwingend einzigartig. Sie setzt sich aus 3 Byte Herstellernummer und 3 Byte Seriennummer zusammen. |
| Malware | Malware beschreibt Bösartige Software / Schadsoftware. Es beschreibt Software welche ausschliesslich dazu entwickelt wurde, schädliche oder böswillige Funktionen auf einem System auszuführen. |
| MTU | die Maximum transmission unit beschreibt die maximale grösse eines Datenpakets welche von einem Empfänger verarbeitet werden kann. Ist die MTU kleiner wie das empfangene Paket so muss das Paket fragmentiert werden und in Teilen übermittelt werden. |

# Abbildungsverzeichnis

## 13.2 Tabellenverzeichnis

## 13.3 Symbolverzeichnis

## 13.4 Abkürzungsverzeichnis

## 13.5 Stichwortverzeichnis

# 14 Anhang

# 15 Projektmanagement

## 15.1 Aufgabenstellung

### 15.1.1 Research-Ziel

1. Analyse der Bluetooth BlueBorne Sicherheitslücken und der Sicherheitspatches
2. Durchführen einer empirische Analyse
3. Suchen nach anfälligen Geräten an der ZHAW oder allenfalls im Raum Winterthur

### 15.1.2 Engineering-Ziel

1. Einarbeiten in die Bluetooth-Architektur
2. Entwickeln eines BlueBorne Testing Modules (Für oder ohne das Malware App)

## 15.2 Zeitplan

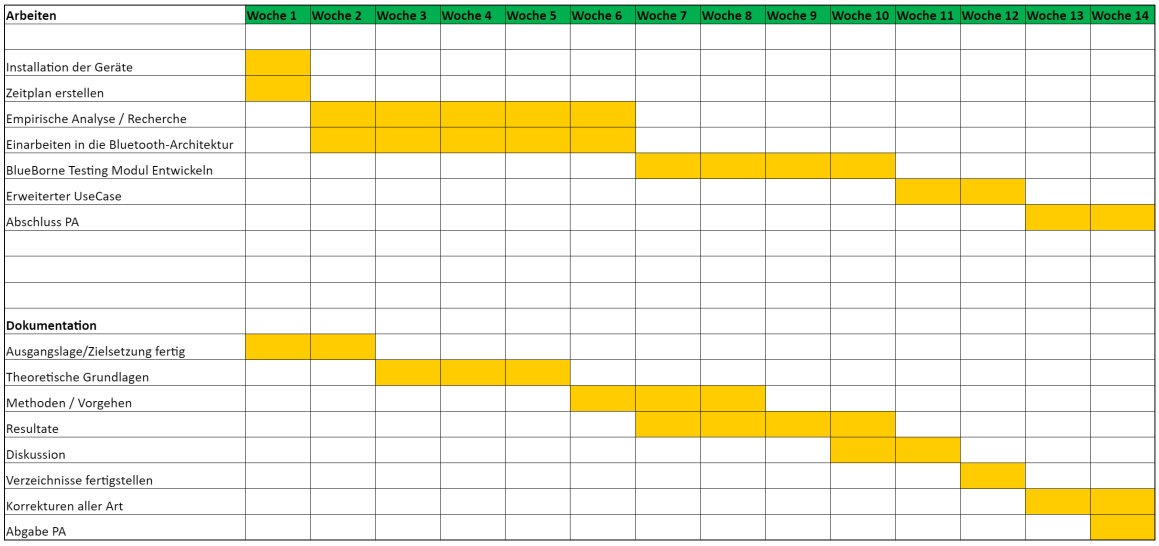


Abbildung 11: Zeitplan der Arbeit und Dokumentation

# 16 Weiteres