Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

InIT Institut für angewandte
Informationstechnologie

Projektarbeit Informatik

BlueBorne War Driving

Autoren	Benjamin Gehring 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. 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. Um die Sicherheitslücken testen zu können werden Geräte benötigt, welche auf einem veralteten Softwarestand betrieben werden können. In dieser Arbeit werden ein Google Nexus 5x[1] und ein Raspberry Pi Model 3b+[2] als Testgeräte verwendet. Die zugrundeliegenden Whitepaper von Armis [4, 3] 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, ein besseres Verständnis der BlueBorne Sicherheitslücke zu erlangen, sowie ein BlueBorne Testing Modul [3]zu entwickeln um potentiell gefährdete Geräte zu erkennen. Dies wird, durch eine vertiefte Einarbeitung in die BlueBorne Thematik mit den korrelierenden CVE Einträgen welche die Angriffe überhaupt ermöglichen, erreicht werden.

5.3 Aufgabenstellung

Die Definierte Aufgabenstellung finden Sie im Punkt 12.1.

[1] Getestet mit verschiedene Versionen von Android 7.1.2 und älter
[2] Installiert mit einem Ubuntu Mate 16.04
[3] Wird später auch als „Scanner Applikation“ bezeichnet

5.4 Anforderungen

Als Ergebniss der Arbeit soll ein Scanner hervorgehen der Android, sowie Linuxgeräte klassifiziert und wenn möglich die Sicherheitslücken teste. Der Scanner versucht zuerst über passives mithören und nachher aktives Anfragen die Geräte in der Umgebung zu lokalisieren und zu bestimmen. Der Scanner ermöglicht einem verschiedene Modi, je nach dem welche Hardware mit dem Scanner in Verwendung ist[4].

[4] Falls ein Ubertooth angeschlossen ist, kann man unsichtbare Geräte entdecken und testen.

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[5].

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 zu 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

[5] 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.

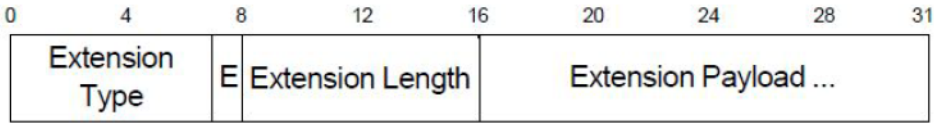

Abbildung 4: Der Extension Header des BNEP Protokolls

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 Vorgehen / Methoden

7.1 Verwendete Hardware

In dieser Arbeit werden ein Nexus 5x Smartphone, ein Raspberry Pi 3B+, ein Ubertooth One und ein CSR Bluetooth Adapter verwendet.
Das **Nexus 5X** [BILD] ermöglicht es beliebige Android Versionen zu installieren. Da der BlueBorne Angriff bereits auf vielen System gepatcht wurde, besteht die Anforderung an eine Möglichkeit zum Downgrade. Ebenfalls sprach für das Nexus 5X, dass dieses bereits in den ofiziellen Skripten von ArmisLab verwendet wurde.
Der **Raspberry Pi 3B+** [BILD] wurde ebenfalls primär wegen den downgrade Möglichkeiten ausgewählt. Von zentraler Bedeutung ist, beliebige Linux Kernel Versionen installieren zu können. Leider existieren keine offiziellen Beispielskripte zu diesem Gerät. ArmisLab selbst demonstrierte den Angriff auf Linux mithilfe eines Amazon Echo One und einer Samsung Tizien Smwartwatch. Es wurde jedoch auf diese Geräte verzichtet, da ein Downgrade nicht, oder nur schwer möglich ist und die zu kaufenden Geräte bereits gepatcht sind.
Der **Ubertooth One** [BILD] ist ein Bluetooth Empfänger und Sender, welcher es ermöglicht versteckte Bluetooth Geräte zu scannen. Ebenfalls kann der Bluetooth Verkehr aufgezeichnet werden.
Der **CSR Bluetooth Adapter** [BILD] ermöglicht das ändern, der eigenen Bluetooth Geräteadresse (BDADDR). Dies ist nötig um ein Zwischenspeichern des Bluetooth Device Names im Speicher des Zielgerätes zu erreichen.

7.2 BlueBorne Angriff auf Android durchführen

Um einen generellen Überblick über das Thema zu erhalten, analysierten wurden in einem ersten Schritt die offiziellen WhitePaper der Entdecker von BlueBorne studiert. Daraus wurde die Erkenntnis gezogen, dass es sich um mehrere, verschiedene Sicherheitslücken handelt. Des weiteren waren Beispielangriffe mit möglichen betroffenen Geräten beschrieben. Dadurch konnten bereits Überlegungen zur benötigten Hardware getätigt und diese beschafft werden. Des weiteren ermöglichte es, eine List, für das Weitere Vorgehen zu erstellen.

1. Angriff auf die Geräte, mithilfe der Beispielskripte, durchführen.
2. Angriff auf weitere Systemversionen ausweiten.
3. Die daraus gewonnen Erkenntnisse verwenden, um eine Applikation zu entwickeln, die gegen diese Angriffe verwundbare Geräte erkennt.

7.2.1 Angriff auf Android v.7.1.2

ArmisLab hat auf GitHub ein Verzeichnis mit Code veröffentlicht[6]. 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, und die Attacke wie beschrieben ausgeführt.

7.2.2 Angriff auf beliebigen Android Versionen

Die zukünftige Scanner App sollte natürlich möglichst alle verwundbaren Android Versionen erkennen. Aus diesem Grund wird der Angriff in einem zweiten Schritt auf weitere Versionen ausgebaut. Ausserdem war dies eine zusätzliche Möglichkeit die Funktionsweise des Beispielskripts nachvollziehen zu können. Da die von Android Version 7.1.2 nachfolgenden Systeme bereits gepatcht wurden, konnte der Angriff nur auf tiefere Versionen ausgebaut werden. Ziel für den nächsten Schritt war also ein erfolgreicher Angriff auf eine tiefere Android Version als 7.1.2.

Durch unsere Recherchen stiessen wir auf einen Blogpost der den Angriff auf Version 6.0.1 beschrieb [1]. Für eine erfolgreiche Umsetzung mussten im Skript vier Variablen angepasst werden, welche die Adressen des Zielgeräts darstellen (BILD). Diese Adressen dienten 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[7] ermöglicht [5]. Daraus konnten anschliessend Offsets berechnet werden. Diese ermöglichen es, mithilfe des Information Leaks beschrieben im Kapitel 6.5, die Basiadresse der Bluetooth 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.

Wie in den Resultaten im Kapitel 8.2 beschrieben, konnten auf der Android Version 6.0.1 nicht alle Variablen ausgelesen werden. Dies war der Grund, weshalb der Versuch auf einer aktuelleren Android Version (v7.1.1), die auch näher an der erfolgreich ausgeführten Version lag, wiederholt wurde. Es wurden die gleichen Anweisungen befolgt, mit der Annahme, dass die gleichen Schritte wie in der Version 6.0.1 benötigt werden würden.

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

7.3 BlueBorne Angriff auf Linux durchführen

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 Stackoverflow Sicherheitslücke intensiver getestet. Sobald der Stackoverflow ohne Stack Canaries durchgeführt werden kann, kann auch der Vollzugriff auf das Gerät weiteren Schritten erfolgen. Diese weiteren Schritte werden für das Verständnis zwar analysiert, jedoch nicht getestet.

7.4 Entwicklung der Scanner Applikation

In einem ersten Schritt wird eine Blueborne Scanner App in Python entwickelt. Diese scannt sichtbare oder unsichtbare Geräte über Bluetooth und sendet eine Payload die 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. 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. Mit diesem Ansatz wird versucht, iterativ mehr über das Gerät zu erfahren und mit gegebenem Wissensstand eine Einschätzung über dieses zu erstellen.

7.4.1 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 der Geräts und wird dem Benutzer angezeigt.

7.4.2 Unsichtbare Geräte - Verbindungsaufbau

Dank dem Ubertooth One können Unsichtbare Geräte entdeckt werden. Der Vorgang wird besser in der Abbildung 7 dargestellt. Wir beziehen uns hierbei auf einen Blogpost von Michael Ossmann [2] einem der Entwickler von Ubertooth und anderen Gadgets.

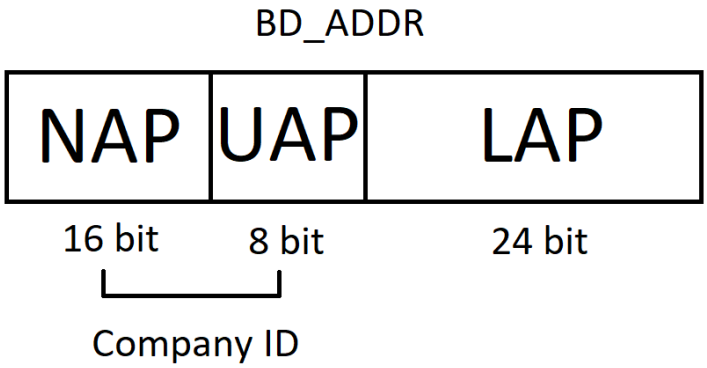

Abbildung 7: Der Aufbau einer Bluetooth Adresse mit eigenen Bezeichnungen

Hierbei ist wichtig zu verstehen, dass nur der LAP Teil der MAC-Adresse, also der „Lower Address Part“ vom Ubertooth erkannt wird. Der UAP, der „Upper Address Part“ kann, sofern man möchte, 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:UAP:LAP:LAP:LAP:LAP“ besitzen[9].
Dieses Vorgehen kann jedoch sehr viel Zeit in Anspruch nehmen, woduch versucht wird, über den Gerätenamen 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.

7.4.3 Klassifizierung der Geräte

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.

[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

7.4.3.1 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.

Abbildung 8: 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.

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 9: Mapping Tabelle für LMP Versionen zu Bluetooth Versionen

[10] Hcitool Dokumentation: https://linux.die.net/man/1/hcitool

8 Resultate

8.1 Angriff auf Nexus 5X mit Android v.7.1.2

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. Dieser Erfolg half uns die Erkenntnisse aus den WhitePaper zu bestätigen und besser zu verstehen wie der Angriff strukturiert wird. Bereits zu diesem Zeitpunkt war es uns möglich erste Überlegungen, über die spätere Scanner Applikation, zu machen.

8.2 Angriff auf Nexus 5X mit Android v.6.0.1

Der verwendete Blogpost [1] bot uns 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. (BILD searchmem funktion ohne resultate?)

8.3 Angriff auf Nexus 5X mit Android v.7.1.1

Auf dieser Version 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, um den Code auszuführen, eine andere war als in der Android Version 7.1.1. 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. Ausserdem waren wir zu der Erkenntnis gelangt, dass für einen erfolgreichen BufferOverflow die Adressen gar nicht nötig waren. Wir konnten diesen also auch auf allen anderen Versionen von Android

(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 dieses Ziel, somit gar nicht nötig.

8.4 Angriff auf Raspberry Pi 3B+

Eifach en Schmööcken.

8.5 Scanner App mit Python

Langsamer Scanner da LAP/UAP Berechnung lange dauert und bnep connections sind auch nicht gerade schnell. Lookup tabelle funktioniert. Weitere klassifizierung.. + zeitmessung beschreiben.

9 Diskussion und Ausblick

9.1 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.

10 Verzeichnisse

Literatur

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

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

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

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

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

10.1 Glossar

Begriff	**Erklärung**
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

10.2 Tabellenverzeichnis

10.3 Symbolverzeichnis

10.4 Abkürzungsverzeichnis

10.5 Stichwortverzeichnis

11 Anhang

12 Projektmanagement

12.1 Aufgabenstellung

12.1.1 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

12.1.2 Engineering-Ziel

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

12.2 Zeitplan

Arbeiten	Woche 1	Woche 2	Woche 3	Woche 4	Woche 5	Woche 6	Woche 7	Woche 8	Woche 9	Woche 10	Woche 11	Woche 12	Woche 13	Woche 14
Installation der Geräte														
Zeitplan erstellen														
Empirische Analyse / Recherche														
Einarbeiten in die Bluetooth-Architektur														
BlueBorne Testing Modul Entwickeln														
Erweiterter UseCase														
Abschluss PA														
Dokumentation														
Ausgangslage/Zielsetzung fertig														
Theoretische Grundlagen														
Methoden / Vorgehen														
Resultate														
Diskussion														
Verzeichnisse fertigstellen														
Korrekturen aller Art														
Abgabe PA														

Abbildung 10: Zeitplan der Arbeit und Dokumentation

13 Weiteres