



**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	4
2	Zusammenfassung	5
3	Abstract	6
4	Vorwort	7
5	Einleitung	8
5.1	Ausgangslage	8
5.2	Zielsetzung	8
5.3	Anforderungen	8
5.4	Aufgabenstellung	9
5.4.1	Research-Ziel	9
5.4.2	Engineering-Ziel	9
6	Theoretische Grundlagen	10
6.1	BlueBorne	10
6.2	Linux Kernel RCE vulnerability - CVE-2017-1000251	10
6.3	BlueZ information leak vulnerability- CVE-2017-1000250	11
6.4	Android information Leak vulnerability - CVE-2017-0785	12
6.5	Android RCE vulnerability Nr. 1 - CVE-2017-0781	13
6.5.1	Schritt 1: ASLR umgehen	14
6.5.2	Schritt 2: Schadcode im Speicher des Opfers platzieren	15
6.5.3	Schritt 3: Schadcode ausführen	15
7	Vorgehen / Methoden	16
7.1	Research	16
7.1.1	Sicherheitslücke auf anderen Android-Versionen ausnutzen	16
7.1.2	Debugging auf Android Geräten	16
7.1.3	Sicherheitslücke in Linux verstehen	16
7.1.3.1	Konzeptuelle Darstellung	17
7.2	Entwicklung	17
8	Resultate	18
9	Diskussion und Ausblick	19
10	Verzeichnisse	20
	Literaturverzeichnis	20
10.1	Glossar	20

Abbildungsverzeichnis	20
10.2 Tabellenverzeichnis	21
10.3 Symbolverzeichnis	21
10.4 Abkürzungsverzeichnis	21
10.5 Stichwortverzeichnis	21
11 Anhang	22
12 Projektmanagement	23
12.1 Zeitplan	23
13 Weiteres	24

1 Formular

Zürcher Hochschule
für Angewandte Wissenschaften



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 Disziplinar massnahmen 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 das Bluetooth Protokoll. Um die Sicherheitslücken testen zu können werden Geräte benötigt, welche auf einem veralteten Softwarestand betrieben werden können. In dieser Arbeit wird ein Google Nexus 5x¹ und ein Raspberry Pi Model 3b+² als Testgeräte verwendet. Die zugrundeliegenden Whitepaper von Armis [2, 1] wurden 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. Des weiteren existiert nur eine sehr begrenzte Anzahl an Scannern, um Geräte auf die beschriebenen Sicherheitslücken zu überprüfen. Alle diese Scanner haben die Gemeinsamkeit, sie überprüfen die Geräte nur nach der MAC-Adresse und nicht nach der installierten Softwareversion. Dies ist zwar weit einfacher, jedoch liefert es auch viel weniger Informationen über das zu testende Gerät.

5.2 Zielsetzung

Das Ziel dieser Arbeit ist es, ein besseres Verständnis der BlueBorne Sicherheitslücke zu erlangen, sowie ein BlueBorne Testing Modul zu entwickeln um potentiell gefährdete Geräte zu erkennen und die Benutzer dieser zu warnen. Dies wird erreicht durch eine vertiefte Einarbeitung in die BlueBorne Thematik mit den korrelierenden CVE³ Einträgen welche die Angriffe überhaupt ermöglichen. 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.

5.3 Anforderungen

Als Ergebniss der Arbeit soll ein Scanner hervorgehen der möglichst viele der 8 BlueBorne Sicherheitslücken auf Geräten, innerhalb eines bestimmten Suchbereichs, erkennt. Der Scanner versucht einen Angriff durchzuführen und meldet bei Erfolg ein entsprechendes vorhandensein der Sicherheitslücke.

¹Getestet mit verschiedene Versionen von Android 7.1.2 und älter

²Installiert mit einem Ubuntu Mate 16.04

³Eine Liste, gefüllt mit Einträgen für öffentlich bekannte Sicherheitslücken

5.4 Aufgabenstellung

5.4.1 Research-Ziel

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

5.4.2 Engineering-Ziel

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

6 Theoretische Grundlagen

6.1 BlueBorne

BlueBorne ist der Überbegriff einiger Sicherheitslücken in der Bluetooth Implementation von Android, IOS, Linux und Windows Geräten. Alle Geräte welche Bluetooth aktiviert haben sind potentiell gefährdet. Die Sicherheitslücke wurde von der Firma Armis entdeckt und im September 2017 öffentlich präsentiert. BlueBorne umfasst total 8 Sicherheitslücken auf verschiedenen Plattformen welche alle zu funktionalen Exploits umgesetzt werden konnten. Alle in dieser Arbeit gezeigten Vorgehen sind stets konzeptuelle Veranschaulichungen.

6.2 Linux Kernel RCE vulnerability - CVE-2017-1000251

Um diese Sicherheitslücke zu misbrauchen 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_NOTRAFFIC`
4. Empfängt: Configuration Request von Opfer mit DCID = 0x0040
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.

6.3 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 eigentlich 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 „lastIndexSent“ 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.

6.4 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_handles Variable

Schafft man es nun, dass number_response_handles 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 baut man eine Verbindung zu einem Service auf, der eine Response schickt die grösser als die angegebene MTU ist.
2. Den dabei erhaltenen Continuation State verwendet man für den zweiten Schritt, wo eine Verbindung zu einem Service aufgebaut wird der eine kleinere Response als die MTU zurückschickt.
3. Innerhalb der Berechnung der „remaining_handles Variable wird nun z.B. 2 - 1 gerechnet. Dies führt bei einem uint16 zu einem Underflow und der Server denkt nun er müsse sehr viele Responses zurückschicken.
4. All diese Responses werden mit einer for-Schleife abgefangen und enthält wichtige Daten des Speichers auf dem Gerät, welche man normalerweise nicht einsehen kann.

Dieser Leak kann in einem weiteren Vorgehen dazu genutzt werden den ASLR Schutzmechanismus gegen BufferOverflows zu umgehen. Dazu berechnet man ausgehend von einer geleakten Memory Adresse einen Offset zurück auf die Basisadresse der Bluetooth Librarys. Während des Angriffs wird ebenfalls dieser Leak genutzt um mithilfe des Offsets die Basisadresse zu berechnen.

6.5 Android RCE vulnerability Nr. 1 - CVE-2017-0781

Die Schwachstellen 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 Mobilfunkverbindungshots-pots über Bluetooth. BNEP fügt dazu einen Header vor der Ethernet Payload an. Zusätzlich werden auch Mechanismen, wie Flusssteuerung über sogenannte Control Messages ermöglicht. Solche Nachrichten sind durch ein Extension Header beschrieben, welcher wiederum mit einem Extension Bit angekündigt wird.

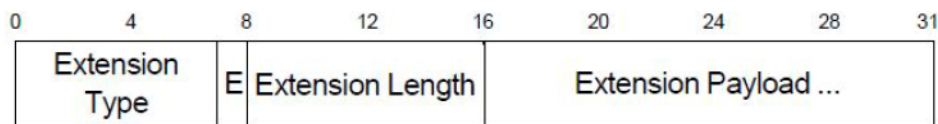


Abbildung 4: Der Extension Header des BNEP Protokolls

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

```

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

Im obigen Codeabschnitt wird eine eingehende BNEP Control Message entgegen-
genommen und verarbeitet. Dabei wird innerhalb des if-Statements die unverarbeitete
Nachricht für die spätere Nutzung in *p_pendig_data* zwischengespeichert. Innerhalb
dieses Vorgangs (memcpy) verbirgt sich allerdings ein entscheidender Fehler: Die un-
verarbeitete 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 Payload als BNEP Nachricht gesendet werden:

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

Abbildung 6: Die Payload welche nötig ist 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 ermöglicht in den korrekten Switch Case zu gelangen.

Für einen erfolgreichen Angriff auf ein Android Gerät müssen noch diverse 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 aus-
geführt werden kann
3. Entsprechender Codeabschnitt überschreiben der diesen Shell Code ausführt

6.5.1 Schritt 1: ASLR umgehen

ASLR sorgt dafür, dass sich die Adressen für Bibliotheken und Services bei jedem
Neustart des Gerätes ändern. Dies macht es natürlich schwierig einen bestimmten
Speicherbereich anzusprechen, wie es im aktuellen Angriff nötig ist. Spezifisch aus-
gedrückt wird die Adresse der Systemfunktion benötigt, um die Programmausführung
auf den Shell Code führen zu lassen und den Speicherort des Bluetooth Devices
Names des Verbindungspartners, um den Shellcode überhaupt im Memory des Op-
fers platzieren zu können. Um diese Ziele zu erreichen hilft der Memory Leak, be-
schrieben im Punkt 6.4, von Android. Die Abstände zwischen einzelnen Speicher-
bereichen sind immer gleich gross. Ziel ist es, die Basisadressen der „libc.so“ Bi-
bliothek und „bluetooth.default.so“ Bibliothek zu ermitteln. Innerhalb „libc.so“ befinden
sich die System Funktionen welche anschliessend, kombiniert mit einem Offset zur
libc.so Basisadresse führen. In der „bluetooth.default.so“ befindet sich der Speicher-
ort des Gerätenamens der später ebenfalls durch einen Offset zur Basisadresse er-
mittelt werden kann. Um die Basisadressen zu berechnen wird im Memory Leak eine

Adresse gesucht die Zwischen Anfangs und Endadresse der „libc.so“ beziehungsweise „bluetooth.default.so“ Bibliothek liegt. Von dieser Adresse wird nun die Basisadresse subtrahiert um den Offset zu erhalten. Bei einem nächsten Neustart muss über den Memory Leak der exakt gleiche Speicherteil nochmals ausgesen werden. Dieser hat nun zwar eine andere absolute Adresse, besitzt jedoch zur Basisadresse noch immer den selben Abstand. Durch diese Gegebenheit kann nun der vorherig berechnete Offset von der ermittelten Adresse subtrahiert werden um die ebenfalls durch den Neustart veränderte Basisadresse zu erhalten. An diesem Punkt ist es wichtig anzumerken dass diese Offsets pro Gerät und Android Version unterschiedlich sind. Für einen erfolgreichen Angriff muss zuerst dieselbe Android Version, wie die des Opfers, auf einem eigenen, identischen Gerät installiert werden, um so die Offsets ermitteln zu können die für den Angriff nötig sind.

6.5.2 Schritt 2: Schadcode im Speicher des Opfers platzieren

Die Platzierung des Schadcodes ist relativ simpel. Bei jedem Bluetooth Verbindungsaufbau wird der Gerätenamen des Verbindungspartners ermittelt und lokal abgelegt. Der Angreifer muss nun seinen Bluetooth Adapter entsprechend des Schadcodes umbenennen. Anschliessend startet er einen Verbindungsaufbau um den Gerätenamen im Speicher des Opfers zu hinterlegen und so den Shellcode zu platzieren. Für einen erfolgreichen Angriff muss nun natürlich noch, wie im ersten Schritt beschrieben, die entsprechende Speicheradresse ermittelt werden.

6.5.3 Schritt 3: Schadcode ausführen

Unter Android 7.1.2 wird mit 80% Wahrscheinlichkeit beim Bufferoverflow ein Zeiger innerhalb einer list_node überschrieben. Ziel ist es nun diesen Zeiger so zu überschreiben, dass er auf den Schadcode zeigt und dieser ausgeführt wird. Solche List Nodes werden bei jedem Verbindungsversuch erzeugt, jedoch teilweise bevor der p_pending_data Buffer alloziert wird. In diesem Fall wäre ein Exploit natürlich nicht möglich. Um zu verhindern, dass die list_nodes vorher alloziert werden, können Löcher zwischen Buffer und list_nodes generiert werden. Die ersten 8 Bytes der Payload überschreiben das Loch mit A's und der zweite Teil der Payload platziert dann die Adresse zum Shell Code in dem list_node.

7 Vorgehen / Methoden

7.1 Research

Als ersten Schritt, analysierten wir die offiziellen WhitePaper der Veröffentlicher von BlueBorne. Um die beschriebenen Erkenntnisse zu bestätigen versuchten wir mithilfe der bereitgestellten Exploit-Programmen der Entdecker von BlueBorne die Attacke nachzubilden.

7.1.1 Sicherheitslücke auf anderen Android-Versionen ausnutzen

Um die Sicherheitslücke zu verstehen wurde unser Google Nexus 5X Testgerät mit einem Android v. 7.1.2 ausgestattet. Bei dieser Androidversion konnten wir sicher gehen dass sie verwundbar ist, da wir Testberichte entdeckt haben, in welchen die gleiche Version verwendet wurde. Nach dem der Exploit auf der vorgegebenen Basis durchgeführt werden konnte, werden wir nach dem gleichen Prinzip auf anderen Android Versionen vorgehen. Dies soll uns dabei helfen ein besseres Verständnis für die Sicherheitslücke zu erhalten, als auch die Hilfswerkzeuge zu verstehen. Des weiteren wird dies als theoretische Grundlage benötigt, um in erweiterten Teilen der Arbeit auch die nötigen Voraussetzungen zu erfüllen um eigene Test-Maleware zu entwickeln.

7.1.2 Debugging auf Android Geräten

Um den Speicher des Telefons zu durchsuchen benötigt man einen Debugger welcher sich an das Telefon anhängen lässt. Mit PEDAS-ARM kann man sich auf einen „gdb-server“, welchen man auf dem Telefon startet, verbinden. Ist man verbunden mit dem debugging Server, so hat man vollen Zugriff auf den aktuellen Speicher des Gerätes sowie debugging Funktionalität wie z.B. Breakpoints. Um den „gdbserver“ auf dem Telefon auführen zu können, muss dieser zuerst auf das Telefon kopiert werden und mit allen Rechten verliehen werden.

7.1.3 Sicherheitslücke in Linux verstehen

Auf dem Raspberry Pi 3b+ wurde eine veraltete Ubuntu Mate Version 16.04 installiert. Diese Version verwendet einen Linux Kernel v. X und Bluetooth(BlueZ) v. X. Beides davon ist genügend alt und somit noch von den beschriebenen Sicherheitslücken im Punkt 6.2 und 6.3 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 nach weiteren Schritten erfolgen.

7.1.3.1 Konzeptuelle Darstellung

Bild von Stackoverflow, Canaries usw.

7.2 Entwicklung

8 Resultate

9 Diskussion und Ausblick

10 Verzeichnisse

Literatur

- [1] B. Seri and A. Livne. Exploiting blueborne in linux based iot devices, 2019.
- [2] B. Seri and G. Vishnepolsky. Blueborne, September 2017.

10.1 Glossar

Begriff	Erklärung
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.
Malware	Malware beschreibt Böartige 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.
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.

Abbildungsverzeichnis

1	CSTATE-Struct	11
2	SDP Search Attribute Request handler	11
3	SDP calculate rem_handels	12
4	BNEP Extension Header	13
5	BNEP BufferOverflow BlueBorne	13
6	BNEP Payload für BufferOverflow	14
7	Zeitplan	23

10.2 Tabellenverzeichnis

10.3 Symbolverzeichnis

10.4 Abkürzungsverzeichnis

10.5 Stichwortverzeichnis

11 Anhang

12 Projektmanagement

12.1 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 7: Zeitplan der Arbeit und Dokumentation

13 Weiteres