Ubertooth - Bluetooth Monitoring und Injection

Martin Herrmann
Betreuer: Stephan Günther
Seminar - Future Internet WS2012/13
Lehrstuhl Netzarchitekturen und Netzdienste (I8)
Fakultät für Informatik, Technische Universität München
Email: martin.herrmann@fs.tum.de

KURZFASSUNG

Es gibt nur wenige Geräte, die Bluetooth *Monitoring* oder *Injection* ermöglichen, da sich dies auf Grund spezieller Eigenschaften des Bluetoothprotokolls als schwierig erweist. Kommerzielle Hardware, welche hierzu in der Lage ist, ist oftmals zu teuer für die Allgemeinheit, weshalb Alternativen wie das *Ubertooth Project* entwickelt werden.
Hierdurch wird es möglich, Angriffe auf Bluetooth Verbindungen durchzuführen, was gravierende Sicherheitslücken aufdeckt. Ferner können Probleme besser adressiert werden, da große Freiheiten in der Entwicklung neuer Anwendungen auf der Basis von Bluetooth eröffnet werden.

Schlüsselworte

Bluetooth, FHSS, Monitoring, Injection, Ubertooth, Authentifizierung

1. EINLEITUNG

Obwohl Bluetoothprodukte bereits seit etwa zehn Jahren erhältlich und sehr weit verbreitet sind, gab es bis vor kurzem keine günstigen, der Allgemeinheit zugänglichen Lösungen, um den Datenfluss aller Bluetoothverbindungen der Umgebung zu überwachen. Ebenso war es nicht möglich, eigene Pakete in bestehende Verbindungen zu injizieren. Diese *Monitoring* und *Injection* genannten Techniken sind für verschiedene Zwecke (z. B. zur Analyse von Verbindungen) von Interesse und in anderen Bereichen, insbesondere bei *IEEE 802.11* (Wireless LAN), längst verfügbar.
Auf Grund dieser Tatsache wurde das *Ubertooth Project* ins Leben gerufen, welches zum Ziel hat, eine offene Plattform für Bluetooth Monitoring und Injection zu schaffen. Im Zuge der Entwicklung entstand der *Ubertooth Zero* und sein Nachfolger, der *Ubertooth One*. Dabei handelt es sich um spezielle Hardware, die in der Lage ist, Daten auf dem 2.4 GHz Band, auf dem auch Bluetooth arbeitet, zu empfangen und zu senden.
Diese Arbeit befasst sich im folgenden zweiten Kapitel mit der Bluetooth Technologie sowie Monitoring und Injection an sich, bevor sie sich im dritten Kapitel mit verschiedenen Monitoring und Injection Lösungen für Bluetooth beschäftigt und diese vergleicht. Im vierten Kapitel werden Anwendungen von Bluetooth Monitorung und Injection vorgestellt, welche auch Angriffe auf fremde Datenübertragungen beinhalten.

2. TECHNISCHE GRUNDLAGEN

2.1 Bluetooth

Bluetooth wurde im Rahmen der IEEE 802.15 Arbeitsgruppe entwickelt, welche sich mit *Wireless Personal Area Networks* (WPAN) beschäftigt [1]. Der Zweck eines WPAN ist es, es verschiedenen Geräten (z. B. Handys oder Laptops) zu ermöglichen, drahtlos miteinander zu kommunizieren.
Der Bluetooth Protokollstapel lässt sich in verschiedene Schichten aufteilen (dargestellt in *Abbildung 1*), von denen für diese Arbeit insbesondere die untersten von Interesse sind, da sie für die Problemstellung des Monitorings wesentlich sind.

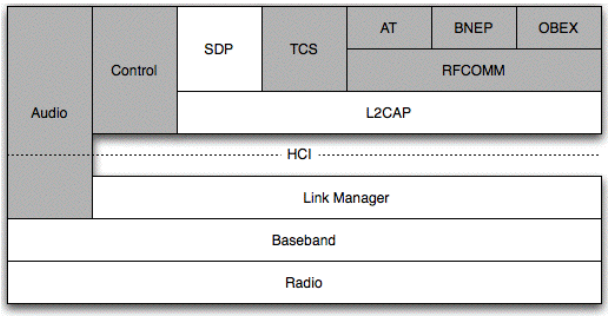

Abbildung 1: Bluetooth Protokollstapel [2]

2.1.1 Radio Layer

Die unterste Schicht ist der so genannte *Radio Layer*. Bluetooth sendet auf dem 2,4 GHz Band, welches zu den ISM Bändern gehört. Da ISM Bänder mit nur wenigen Einschränkungen von jedem genutzt werden können, kann nicht garantiert werden, dass das Signal störungsfrei übertragen wird. Deswegen kommt hier ein *Frequency Hopping Spread Spectrum* (FHSS) genanntes Verfahren zum Einsatz (siehe *Abbildung 2*), welches den Frequenzbereich zwischen 2,4 GHz und 2,4835 GHz in 1 MHz breite Kanäle unterteilt [1]. Außerdem existiert ein 3 MHz breiter Schutzabstand am oberen Rand des Spektrums, sowie ein 1,5 MHz breiter Abstand am unteren Rand, weshalb sich 79 nutzbare Kanäle ergeben, zwischen denen alle 625 µs gewechselt wird. Wenn auf einer bestimmten Frequenz also Störungen vorliegen (z. B. durch parallele Nutzung von anderen Geräten), kommt es nur kurzzeitig zu Übertragungsproblemen, da der Kanal sofort wieder gewechselt wird. Ferner hat dieses Verfahren den Vorteil, dass das gezielte Abhören einer Verbindung erschwert wird, da ein Angreifer die verwendete Sprungfolge kennen muss.

doi: 10.2312/NET-2013-02-1_03

Als Modulationsverfahren wird *Gaussian Frequency Shift Keying* (GFSK) verwendet, welches zusätzlich zum eigentlichen modulieren der Frequenz auch noch einen Gauss-Filter auf das Signal anwendet.

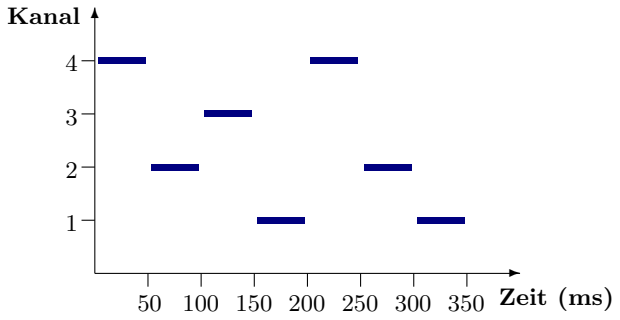

Abbildung 2: Einfaches Beispiel für ein FHSS Verfahren (4 Kanäle, Sprung alle 50 ms)

2.1.2 Baseband Layer

Der *Baseband Layer* verwaltet die physikalischen Verbindungen. Jedes Gerät kann hier über eine 48 Bit lange *Bluetooth Device Address* (BD_ADDR) eindeutig identifiziert werden. Diese kann in *Lower Address Part* (LAP, 24 Bit), *Upper Address Part* (UAP, 8 Bit) und *Nonsignificant Address Part* (NAP, 16 Bit) unterteilt werden (siehe *Abbildung 3*), wobei UAP und NAP zusammen den Hersteller identifizieren, während der LAP von diesem vergeben wird. Für bestimmte Zwecke existieren die reservierten LAPs *0x9E8B00–0x9E8B3F*. Für diese wird 0x00 als UAP verwendet.

16 Bit	8 Bit	24 Bit
NAP	**UAP**	**LAP**

Abbildung 3: Bluetooth Device Address Format [1]

Eine physikalische Verbindungen (als *Piconet* bezeichnet) besteht aus einem Master und mindestens einem Slave. Adressiert wird sie durch die Verwendung eines *Access Code*, welcher aus dem LAP des Masters generiert wird. Damit die Datenübertragung möglich ist, müssen natürlich alle Teilnehmer die gleiche Sprungfolge für die Frequenzen verwenden. Diese wird aus LAP und UAP des Masters errechnet, und ist in etwa gleichverteilt über die 79 Kanäle. Da sich der Master und die Slaves auch an der gleichen Stelle in der Sprungfolge befinden müssen, werden die Slaves zusätzlich auch noch zeitlich mit dem Master synchronisiert [1].

2.1.3 Paketaufbau

Alle Pakete werden nach dem selben Schema aufgebaut (siehe *Abbildung 4*). Sie beginnen immer mit dem *Access Code*, gefolgt von dem *Header* und der *Payload*. Es gibt Pakete, die nur den *Access Code* enthalten, welche die nur den *Access Code* und den *Header* enthalten, und solche, die alle drei Teile beinhalten [1]. Der *Access Code* besteht aus einer 4 Bit Präambel, gefolgt von einem 64 Bit langen *Sync Word*, welches aus LAP und UAP berechnet wird. Falls ein Header folgt, endet der *Access Code* noch mit einem 4 Bit Trailer. Der Header enthält Felder die zur Steuerung des Datenflusses benutzt werden. Dazu gehört zum Beispiel ein Feld, welches den Pakettyp spezifiziert, sowie Felder zur Übertragungskontrolle und zur Adressierung einzelner Slaves.

68/72 Bit	54 Bit	0 – 2745 Bit
Access Code	**Header**	**Payload**

Abbildung 4: Paketaufbau [1]

Eines der einfachsten Pakete ist das ID Paket. Es besteht nur aus dem *Access Code* und hat daher eine Länge von 68 Bit. Für die Berechnung des Sync Word wird die für *Inquiries* reservierte LAP *0x9E8B33* benutzt.
Ein sehr wichtiges Paket für den Verbindungsaufbau ist das *Frequency Hopping Synchronization* (FHS) Paket. Es enthält zusätzlich zum *Access Code*, welcher dieses mal aus dem LAP und UAP eines anderen Gerätes berechnet wurde und somit an dieses adressiert ist, einen Header sowie Payload. Die Payload enthält unter anderem alle drei Teile der BD_ADDR, sowie die interne CLK des Gerätes. Dieses Paket enthält alle Informationen, die zur Berechnung der Sprungfolge benötigt werden.
Ferner existieren noch eine Reihe anderer Pakete, insbesondere solche zur Datenübertragung. Dabei gibt es verschiedene Typen für spezielle Zwecke, von denen einige auch größer als ein Übertragungsslot sind (z. B. 3 oder 5 Slots).

2.1.4 Verbindungsaufbau

Für den Verbindungsaufbau müssen die Geräte zunächst gepaart werden. Dazu schickt der zukünftige Master *Inquiries* aus (ID Pakete). In diesem Fall werden nur 32 der 79 möglichen Kanäle für die Sprungfolge verwendet. Geräte, die auf *Sichtbar* gestellt sind, antworten mit einer *Inquiry Response*, welche bereits die Adresse und die CLK des Gerätes enthält. Um das *Pairing* fortzuführen, berechnet der Master aus der erhaltenen Adresse eine spezifische Sprungfolge und synchronisiert sie mit dem Takt des Slaves, der sich nun im *Page Scan* Modus befindet. Diese Sprungfolge benutzt der Master nun, um dem Slave die nötigen Informationen mitzuteilen, die er für das Errechnen der entgültigen Sprungfolge braucht. Er sendet dazu zwei ID Pakete und wartet im nächsten Sendeslot auf ein ID Paket des Slaves als Antwort. Nun sendet der Master ein FHS Paket, welches der Slave mit einem weiteren ID Paket beantwortet. Da der Slave nun die korrekte Sprungfolge kennt, wechselt der Master nun zu dieser. Grundsätzlich sendet der Master hierbei immer in den geraden Slots der Sprungfolge, während der Slave in den ungeraden sendet.

2.1.5 Sicherheit

Bluetooth verfügt auch über Sicherheitsverfahren, welche in Kapitel 13 der Bluetooth Spezifikationen detailliert beschrieben werden [1]. Wenn ein physikalischer Kanal (synchronisierte Frequenz Sprungfolge) exisitiert, werden über ihn logische Verbindungen aufgebaut. Dabei müssen in der Regel

doi: 10.2312/NET-2013-02-1_03

die PINs auf den Geräten übereinstimmen oder es muss der korrekte PIN für das entfernte Gerät eingegeben werden, falls dieser fest ist (z. B. bei Headsets ohne Interface).
Um die Authentizität einer Verbindung sicherzustellen, wird zuerst der K_{init} Schlüssel generiert. Dazu wird der E_{22} Algorithmus (welcher hier nicht genauer erklärt wird), mit dem verwendeten PIN, einer 128 Bit langen Zufallszahl *IN_RAND* (von Master generiert, an Slave übertragen) und der *BD_ADDR* des Slaves (bei festem PIN des Slaves die des Masters) als Parameter, ausgeführt.
Der K_{init} Schlüssel wird zur Verschlüsselung der Übertragung von zwei weiteren 128 Bit Zufallszahlen (LK_RAND_A und LK_RAND_B) verwendet, welche zusammen mit den BT_ADDR der beiden Geräte zur Berechnung des *Link Key*, welcher nun zur Authentifizierung benutzt werden kann.
Hierfür schicken sich die beiden Geräte wieder gegenseitig 128 Bit lange Zufallszahlen AU_RAND_A und AU_RAND_B, welche dann mit dem zuvor gewonnenem Schlüssel verschlüsselt werden. Die Ergebnisse ($SRES_A$ und $SRES_B$) der Operationen auf beiden Geräten werden verglichen und die Verbindung wird nur zugelassen, wenn sie übereinstimmen [3].
Der Schlüssel wird solange gespeichert, bis das *Pairing* aufgehoben wird. Ferner kann aus diesem Schlüssel ein weiterer generiert werden, der zur Verschlüsselung der Datenübertragung genutzt wird. Dieser hat eine Länge von 8 – 128 Bit.

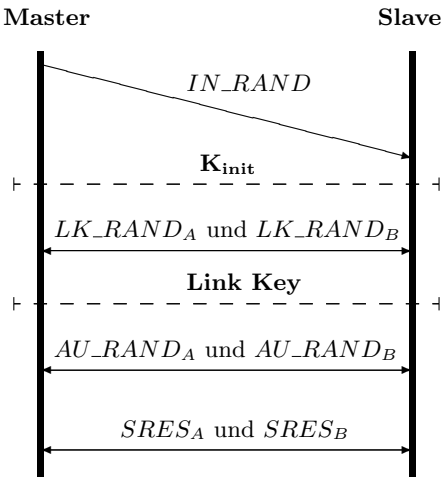

Abbildung 5: Ablauf des Authentifizierungsprozesses

2.2 Monitoring und Injection

Da ein Kanal nur selten exklusiv benutzt wird, kommen immer auch Daten an, welche für andere Empfänger bestimmt waren. Damit diese nicht die eigene Kommunikation stören, wird im normalen Betriebsmodus beim Erhalt eines neuen Pakets zunächst überprüft, ob es für dieses Gerät bestimmt ist. Im Falle von (Wireless) LAN werden hierzu die *MAC-* und die *IP-Adressen* überprüft, während bei Bluetooth der *Access Code* ausschlaggebend ist. Pakete, die nicht für ein bestimmtes Gerät bestimmt sind, werden von diesem ignoriert und nicht an höhere Schichten weitergeleitet.
Manchmal ist es allerdings von Interesse, auch solche Pakete zu erhalten, die nicht für einen selbst bestimmt sind (z. B. zur Überwachung von Netzwerken). In diesem Fall werden alle Pakete, die von der Netzwerkhardware empfangen werden, an höhere Schichten weitergeleitet, was man auch als *Monitoring* oder *Monitor Mode* der Hardware bezeichnet. Die empfangenen Daten werden dann in der Regel von einem Programm wie *Wireshark* [4] oder *Kismet* [5] verarbeitet und dargestellt. *Abbildung 6* zeigt Wireshark, das dank *Monitor Mode* Funktionalität des WLAN-Adapters alle in der Umgebung verschickten Pakete anzeigt.

Destination	Protocol	Length	Info
Wistron_29:43:03	802.11	1583	QoS Data, SN=565, FN=0, Flags=.p....F.C
Wistron_29:43:03	802.11	1583	QoS Data, SN=566, FN=0, Flags=.p....F.C
Wistron_29:43:03	802.11	1583	QoS Data, SN=567, FN=0, Flags=.p....F.C
Wistron_29:43:03	802.11	1583	QoS Data, SN=568, FN=0, Flags=.p....F.C
Wistron_29:43:03	802.11	1583	QoS Data, SN=569, FN=0, Flags=.p....F.C
Wistron_29:43:03	802.11	1583	QoS Data, SN=570, FN=0, Flags=.p....F.C
Wistron_29:43:03	802.11	1583	QoS Data, SN=571, FN=0, Flags=.p....F.C
Tp-LinkT_f3:72:1a (RA	802.11	58	802.11 Block Ack, Flags=........C
Tp-LinkT_f3:72:1a (RA	802.11	46	Request-to-send, Flags=........C
Wistron_29:43:03 (RA)	802.11	40	Clear-to-send, Flags=........C
Tp-LinkT_f3:72:1a	802.11	135	QoS Data, SN=32, FN=0, Flags=.p.....TC

Abbildung 6: Monitor Mode und Wireshark

Wenn Daten gesendet werden sollen, wird in der Regel eine von der Netzwerkhardware (bzw. von ihrem Treiber) bereitgestellte Schnittstelle verwendet. Üblicherweise muss sich die Anwendung nicht um den genauen Ablauf der Kommunikation kümmern, die Übertragung erfolgt also transparent. Für gewisse Anwendungen ist es allerdings vonnöten, tiefer in den Kommunikationsablauf einzugreifen. So gibt es die *Injection* Technik, bei der eigene Datenpakete in fremde Verbindungen eingeschleust werden, um so z. B. Verschlüsselungen zu knacken oder *Spoofing* Angriffe auszuführen. Dies muss natürlich auch von der Hardware bzw. vom Treiber unterstützt werden, da hier die normalen Senderoutinen umgangen werden. Eine sehr bekannte Anwendung ist hier das Knacken der WEP-Verschlüsselung mittels der *Aircrack-ng Suite* [6], welche auch die Bedeutung der Verfügbarkeit dieser Techniken aufzeigt.
Selbst wenn gewisse Verfahren (wobei hier WEP als Paradebeispiel dient) in der Theorie längst nicht mehr sicher sind, werden sie trotzdem weiterhin eingesetzt, sofern sich die praktische Umsetzung dieser Angriffe schwierig gestaltet. Der Gedanke dahinter ist oftmals, dass es die Allgemeinheit nicht betrifft, was dazu führt, dass sich einem kleineren Personenkreis weite Angriffsmöglichkeiten eröffnen. Wenn theoretische Angriffe allerdings für jedermann leicht ausführbar werden, wie dies bei WEP durch die Verbreitung von *Monitoring* und *Injection* fähiger Hardware sowie entsprechender Tools geschah, ist man gezwungen sichere Verfahren zu entwickeln und auch einzusetzen, was einen erheblicher Gewinn darstellt.

3. MONITORING UND INJECTION LÖSUNGEN FÜR BLUETOOTH

Wie bereits in der Einleitung erwähnt, gab es für Bluetooth lange Zeit kaum Geräte, welche über einen *Monitor Mode* oder die Möglichkeit der *Packet Injection* verfügten.
Hindernisse hierfür hierfür werden in Abschnitt 3.1 diskutiert. In Abschnitt 3.2 werden einige professionelle Lösungen vorgestellt, bevor sich Abschnitt 3.3 mit *Ubertooth* im Detail beschäftigt. Zum Abschluss werden in Abschnitt 3.4 die verschiedenen Lösungen miteinander verglichen.

doi: 10.2312/NET-2013-02-1_03

3.1 Gründe für den Mangel

Das erste Hindernis beim Abhören von Bluetoothverbindungen ist die Verwendung des FHSS Verfahrens (siehe *Abschnitt 2.1.1*). Um eine komplette Datenübertragung zu empfangen, muss man entweder alle 79 Kanäle gleichzeitig abhören und die richtigen Pakete herausfiltern, oder die verwendete Sprungfolge kennen. Ersteres setzt hohe Ansprüche an die verwendete Hard- und Software, für die zweite Möglichkeit muss man warten, bis man Pakete empfängt und daraus versuchen, die Sprungfolge zu rekonstruieren.
Ein weiteres Problem ist die Filterung der eingehenden Pakete auf Grund ihres *Access Code*. Da dies in gängigen Bluetoothsticks laut dem Ubertooth Entwickler Michael Ossmann [7] meist direkt auf Hardware-Ebene geschieht, gestaltet es sich mit diesen schwierig, alle empfangenen Pakete an den PC weiterzuleiten. Ferner liegt der Gedanke nahe, dass die Hersteller kein Interesse daran haben, entsprechende Funktionalitäten zu implementieren, da sie so eine gewisse *Security by obscurity* aufrecht erhalten können.

3.2 Professionelle Lösungen

Trotz der im vorherigen Abschnitt beschriebenen Probleme existiert kommerzielle Analyse-Hardware für Bluetooth, wie der *Ellisys Bluetooth Explorer 400*. Laut Herstellerinformationen [8] ist dieser ist in der Lage alle 79 Bluetooth-Kanäle gleichzeitig zu überwachen, was es auch möglich macht, alle Piconetze der Umgebung gleichzeitig zu beobachten. Ferner untersützt er alle neueren Bluetooth Standards wie *Enhanced Data Rate* (EDR) und *Low Energy* (LE) und ist in der Lage, den verwendeten PIN einer Bluetoothübertragung zu berechnen und damit die Verschlüsselung zu knacken (mehr hierzu in *Kapitel 4*), sofern ein Pairing-Prozess beobachtet wird. Die Hardware wird mittels einem USB 2.0 Anschluss an einen PC angeschlossen, entsprechende Software ist im Lieferumfang. Eine Anfrage hat ergeben, dass sich der Preis für ein komplettes System um die 20 000 US Dollar bewegt.
Eine weitere Bluetooth-Monitoring Lösung ist der *ComProbe BPA500*. Dieser unterstützt laut Hersteller [9] ebenfalls EDR und LE und kann bei Beobachtung eines Pairing Prozesses den verwendeten PIN errechnen. Ferner lässt sich die Hardware noch um weitere Funktionalitäten (wie zusätzliches WLAN Monitoring) erweitern. Eine Preisanfrage wurde noch nicht beantwortet, die Kosten sind aber wahrscheinlich mit denen des Bluetooth Explorers vergleichbar.

3.3 Ubertooth

Im Gegensatz zu der in *Abschnitt 3.2* beschriebenen Lösungen liegt das Ziel des *Ubertooth Project* nicht in der Schaffung eines fertigen, kommerziellen Endprodukts, sondern der Entwicklung einer open-source Plattform, welche von jedem verwendet werden kann, der er über grundlegende Kenntnisse im Bereich der Netzwerktechnik verfügt [10]. Die Kosten für einen *Ubertooth One* liegen bei ca. 100 US Dollar.

Für die Hardware (siehe *Abbildung 7*) wird eine 4-lagige Leiterplatte verwendet. Es existiert ein *RP-SMA* Anschluss [12] für eine Antenne. Ein *CC2591 RF Front End* von Texas Instruments verarbeitet das analoge Signal für einen *CC2400 Wireless Transceiver*, welcher im 2.4 – 2.4835 GHz Bereich empfangen und senden kann [13][14]. Das Signal wird von einem Mikroprozessor (*Arm Cortex-M3* Architektur) der *LPC175x* Serie verarbeitet. Die vorliegende Hardware arbeitet mit einem *LPC1756*, welcher über 256 kB Flash Memory für die Firmware verfügt. Er arbeitet bei einer Taktfrequenz von maximal 100 MHz und wird mit einem USB 2.0 Anschluss verbunden [11].

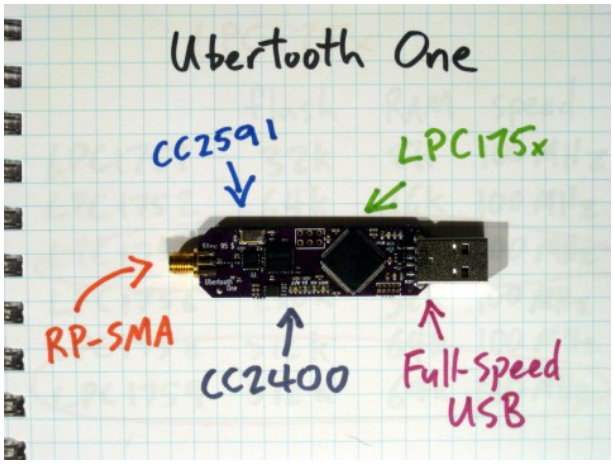

Abbildung 7: Aufbau des *Ubertooth One* [11]

Aus dieser Hardwarekonfiguration ergibt sich auch direkt eine Beschränkung: der CC2400 kann maximal mit 1 Mbit/s senden und empfangen [13]. Dies führt dazu, dass neuere Bluetooth Standards ab der Version 2.0 nicht komplett unterstützt werden können, da diese sich unter anderem durch *Enhanced Data Rate* (EDR) auszeichnen, welche bis zu 2,1 Mbit/s erreichen kann.
Um den *Ubertooth One* zu verwenden, muss man zunächst eine aktuelle Version der Firmware installieren, was am einfachsten mit der vorhandenen USB DFU Utility ist. Der auf der Projektseite verfügbare Quellcode enthält bereits einige Tools, die der Reihe nach vorgestellt werden. Diese funktionieren problemlos mit aktuellen Linux-Systemen, sofern einige zusätzliche Pakete installiert werden. Eine Portierung auf andere Systeme (z. B. MAC oder Windows) gestaltet sich problemlos.
Mit Hilfe des *Specan UI* Tools kann man sich den Signalstärkeverlauf des Frequenzbereichs in Echtzeit graphisch anzeigen. Da auch WirelessLAN in diesem Bereich sendet, kann man auch dieses so beobachten. So erkennt man auf *Abbildung 8* beispielsweise ein aktives WLAN um 2,415 MHz. Alternativ zu *Specan UI* kann man auch *ubertooth-specan* benutzen, welches die Rohdaten in einem für entsprechende Analysesoftware geeignetem Format ausgibt.

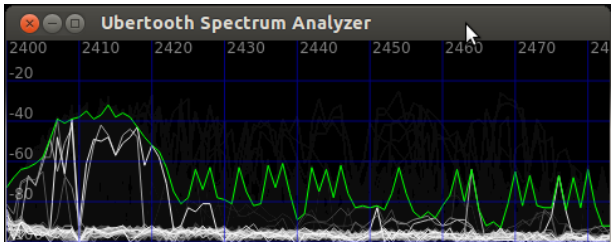

Abbildung 8: Signalstärkeverlauf mit *Specan UI*

Mit *ubertooth-util* lassen sich die Betriebsmodi des Gerätes

doi: 10.2312/NET-2013-02-1_03

konfigurieren. Hier kann man zum Beispiel den zu überwachenden Kanal auswählen, das Gerät in den *Device Firmware Upgrade* Modus versetzen oder sich diverse Informationen (wie die aktuelle Firmware Version) anzeigen lassen.
Das Tool *ubertooth-lap* ermöglicht Monitoring auf einem spezifizierten Kanal und gibt den LAP aus. Mittels *ubertooth-uap* lässt sich zu einem gegebenen LAP auch der dazugehörige UAP bestimmen. Obwohl hiermit schon eine einfache Beobachtung von Bluetoothverbindungen möglich sind, lässt sich dies noch besser mit dem Kismet-Plugin für Ubertooth realisieren. Dies bestimmt sowohl den LAP als auch den UAP der empfangen Pakete und kann diese als *pcapbtbb* Dumpfile ausgeben, die man dann in anderer Software weiter analysieren kann (siehe *Abbildung 9*).

Time	Source	Destination	Protocol	Length	Info
11.749314	00:00:00_c0:36:a6	00:00:00_00:00:00	Bluetooth	14	ID
12.258369	00:00:00_c0:36:a6	00:00:00_00:00:00	Bluetooth	14	ID
12.626241	00:00:00_14:69:bb	00:00:00_00:00:00	Bluetooth	14	ID
12.658602	00:00:00_14:69:bb	00:00:00_00:00:00	Bluetooth	14	ID
12.869856	Camex_c0:36:a6	00:00:00_00:00:00	LMP	27	LMP_setup_complete
12.908252	Camex_c0:36:a6	00:00:00_00:00:00	Bluetooth	51	DH1/2-DH1
12.911112	Camex_c0:36:a6	00:00:00_00:00:00	LMP	38	LMP_accepted
13.234873	Camex_c0:36:a6	00:00:00_00:00:00	Bluetooth	23	NULL
13.263188	Camex_c0:36:a6	00:00:00_00:00:00	Bluetooth	23	POLL
13.263595	Camex_c0:36:a6	00:00:00_00:00:00	Bluetooth	23	POLL
13.461768	Camex_c0:36:a6	00:00:00_00:00:00	Bluetooth	23	NULL
13.490820	Camex_c0:36:a6	00:00:00_00:00:00	Bluetooth	23	POLL

Abbildung 9: Bluetoothpakete in Wireshark

Leider scheint das Verschicken von Paketen über die USB-Schnittstelle bisher noch nicht möglich zu sein. Es gibt allerdings die Möglichkeit mit zwei Ubertooth Sticks einen Testmodus zu starten, bei dem sie sich gegenseitig Bluetooth ähnliche Pakete zusenden, was darauf hindeutet, dass die Firmware in absehbarer Zeit auch *Packet Injection* unterstützen wird.
Zusammenfassend lässt sich sagen, dass der *Ubertooth One* eine sehr mächtige Plattform für Bluetooth Monitoring und Injection darstellt, auch wenn viele Funktionalitäten (noch) nicht vorhanden sind. Ferner kann man die Ubertooth Plattform (entsprechende Firmware vorausgesetzt) auch für andere Systeme, welche im gleichen Frequenzbereich arbeiten, verwenden.

3.4 Vergleich

Da es sich bei den in *Abschnitt 3.4* vorgestellten Lösungen um kommerzielle Bluetooth-Monitoring Hardware handelt, kann der *Ubertooth One* natürlich nicht komplett mithalten, da diesem zum Beispiel (wie bereits in *Abschnitt 3.3* erwähnt) die EDR-Funktionalität fehlt. Außerdem befindet sich das Ubertooth-Projekt noch in der Entwicklungsphase, was bedeutet, dass viele Funktionalitäten noch nicht komplett umgesetzt sind. All dies bedeutet, dass der *Ubertooth One* im Moment noch nicht für professionelles Bluetooth-Monitoring, wie dies zum Beispiel bei der Entwicklung neuer Hard- und Software nötig ist, geeignet ist.
Dennoch ist er gerade für akademische Zwecke sehr gut geeignet, da es hier unter anderem auf geringe Kosten ankommt. Dass das Projekt quelloffen ist, ist ein weiterer Vorteil, da es somit sehr gut an spezielle Bedürfnisse angepasst werden kann, ohne dass man in langwierige Verhandlungen mit den Herstellern treten muss.

4. ANWENDUNGEN

Nachdem in *Kapitel 3* nun einige Möglichkeiten des Bluetooth Monitorings vorgestellt wurden, werden hier nun einige Anwendungen beschrieben, die über die bloße Beobachtung, beispielsweise zu Debugging-Zwecken, hinausgehen.

4.1 Knacken der Authentifizierung

In ihrer Arbeit [3] haben Yaniv Shaked und Avishai Wool ein Bruteforce Verfahren beschrieben, welches in der Lage ist den PIN zu berechnen, falls das *Pairing* beobachtet wird. Neben allgemein bekannter Informationen wie die Adressen der Geräte sind hierzu sind die Zufallszahlen IN_RAND, LK_RAND_A und LK_RAND_B (jeweils mit K_{init} verschlüsselt), AU_RAND_A und AU_RAND_B, sowie die Ergebnisse $SRES_A$ und $SRES_B$ nötig.
Man beginnt mit einer Annahme für den PIN (in der Regel 0) und berechnet hierzu den K_{init}, mit dem man dann das erste Paar an Zufallszahlen entschlüsselt. Aus diesen kann man wiederum den *Link Key* berechnen, um mit ihm das zweite Zufallsvariablenpaar zu verschlüsseln. Stimmt dieses mit den Ergebnissen $SRES_A$ und $SRES_B$ überein, so verfügt man über den verwendeten PIN (siehe *Abbildung 10*), andernfalls wiederholt man den Prozess mit einem weiteren PIN.

Da sowohl die Authentifizierung zwischen den Geräten, als auch die Verschlüsselung der Verbindung auf dem PIN aufbaut, ist es nun ein leichtes verschlüsselte Übertragungen zu belauschen oder sich gar als einer der Teilnehmer auszugeben. Ferner können diese Berechnungen sehr effizient implementiert werden, so dass man kurze PINs auf moderner Hardware innerhalb weniger Sekunden knacken kann. Demonstriert wird dies unter anderem durch das Tool *BT-Crack* [15], welches genau nach diesem Prinzip arbeitet und frei zugänglich ist.

4.2 Erzwingen eines erneuten Pairings

Da Geräte einen einmal berechneten *Link Key* in der Regel auf Dauer abspeichern, ist es nicht immer möglich, ein *Pairing* zu beobachten. Dies ist allerdings für den zuvor beschriebenen Angriff nötig, da man nur so in den Besitz der benötigten Informationen gelangen kann. Hierfür kann man allerdings die Tatsache ausnutzen, dass es möglich ist, den *Link Key* zu löschen. Dies ist zum Beispiel der Fall, wenn der Benutzer das *Pairing* an seinem Gerät löst [1].
Wenn nun das andere Gerät versucht, eine Verbindung aufzubauen, wird es wie üblich eine AU_RAND Zufallszahl schicken, nur das dieses mal ein *LMP_not_accepted* Paket als Antwort hierauf geschickt wird. Wenn ein Angreifer nun also zumindest einen Verbindungsaufbau beobachten kann und zusätzlich in der Lage ist, eine *Packet Injection* durchzuführen, kann er dem eigentlichen Slave zuvorkommen, und *LMP_not_accepted* schicken, was das *Pairing* löst. Des weiteren kann ein Angreifer auch ein falsches $SRES$ injizieren oder ein IN_RAND Paket schicken, bevor die AU_RAND Zufallszahl gesendet wurde [3].
Alle drei beschriebenen Techniken haben zur Folge, dass die Geräte erneut ein *Pairing* durchlaufen, und man den eigentlichen Angriff auf den PIN starten kann.

doi: 10.2312/NET-2013-02-1_03

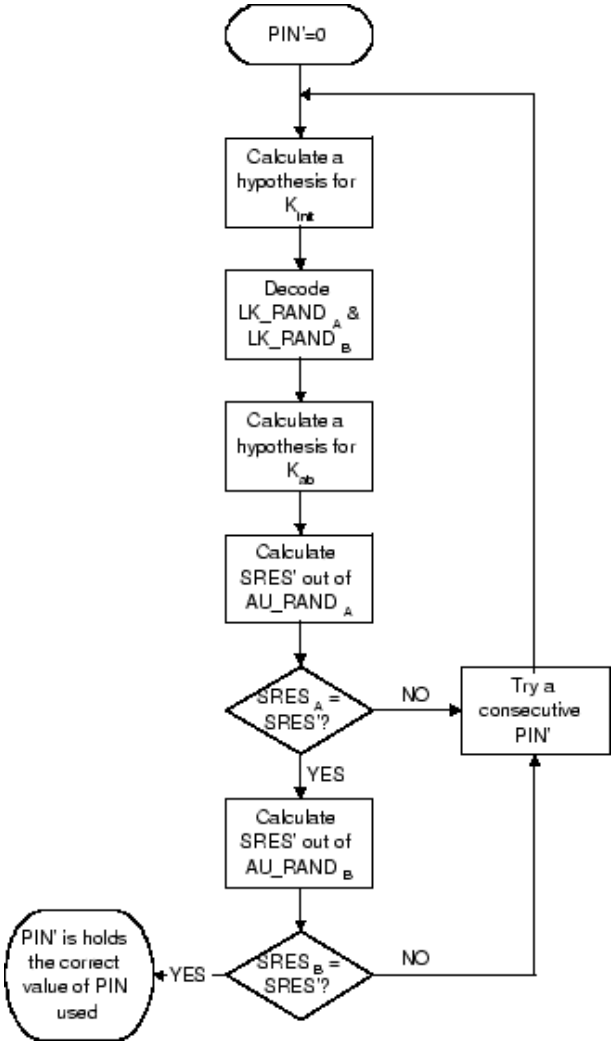

Abbildung 10: Ablauf des Angriffes [3]

4.3 Modifizierungen des Bluetoothprotokolls

Für bestimmte Anwendungen kann es von Interesse sein, das Bluetoothprotokoll in leicht modifizierter Version anzuwenden. Da Bluetooth für eine große Anzahl an Anwendungsmöglichkeiten entwickelt wurde, ist es nicht immer optimal für einen bestimmten Zweck.
So können Sicherheitsfunktionen auf höheren Schichten implementiert worden sein, welche eine erneute Authentifizierung und Verschlüsselung auf niedrigeren Schichten obsolet machen. Außerdem kann es ausreichend (oder sogar von Vorteil) sein, stets eine feste Sprungfolge zu verwenden, so dass es eine Reihe von Systemen komplett störungsfrei nebeneinander betrieben werden können.
Da solche Modifizierungen tiefer in die Abläufe eingreifen, braucht man spezielle Hardware, die unter anderem *Monitoring* und *Injection* unterstützen muss.

5. ZUSAMMENFASSUNG

Es besteht ein Bedarf an Bluetooth *Monitoring* und *Injection* Lösungen, welche auch für die Allgemeinheit zugänglich sind. Professionelle Hardware erfüllt zwar alle technischen Anforderungen problemlos, ist aber auf Grund ihrer sehr hohen Kosten hierfür nicht geeignet. Das *Ubertooth Project* ist hier eine sehr viel versprechende Plattform, insbesondere da sie (trotz gewisser Einschränkungen) sehr gut auch an speziellere Bedürfnisse angepasst werden kann und gleichzeitig sehr günstig ist. Leider kann mit bisher die Möglichkeiten der Hardware noch nicht voll ausnutzen, weshalb zu hoffen bleibt, dass die Entwicklung zügig voranschreitet.
Insbesondere auf die Sicherheit von Bluetooth dürfte sich die Weiterentwicklung des *Ubertooth Projects* positiv auswirken, da hier große Schwächen vorliegen, welche durch die Verbreitung von Bluetooth Monitoring und Injection stark in den Vordergrund rücken werden.

6. LITERATUR

[1] IEEE Computer Society. IEEE Standard 802.15.1-2005, 2005.

[2] Greg Hackmann. 802.15 Personal Area Networks. `http://www.cse.wustl.edu/~jain/cse574-06/ftp/wpans/index.html`, 2006. [Online; Stand 23. September 2012].

[3] Yaniv Shaked and Avishai Wool. Cracking the Bluetooth PIN. In *in Proc. 3rd USENIX/ACM Conf. Mobile Systems, Applications, and Services (MobiSys*, pages 39–50, 2005.

[4] Wireshark. `http://www.wireshark.org/`, 2012. [Online; Stand 23. September 2012].

[5] Kismet. `http://www.kismetwireless.net/`, 2012. [Online; Stand 23. September 2012].

[6] Aircrack-ng. `http://www.aircrack-ng.org/`, 2012. [Online; Stand 23. September 2012].

[7] Michael Ossmann. ShmooCon 2011: Project Ubertooth: Building a Better Bluetooth Adapter. `http://www.youtube.com/watch?v=KSd_1FE6z4Y`, 2011. [Online; Stand 23. September 2012].

[8] Ellisys. Ellisys Bluetooth Explorer 400 Datasheet. `http://www.ellisys.com/products/download/bex400_brochure.pdf`, 2010.

[9] Frontline Test Equipment. ComProbe BPA500 Datasheet. `http://www.fte.com/docs/datasheet_bpa500.pdf`, 2012.

[10] Michael Ossmann. Ubertooth Project. `http://ubertooth.sourceforge.net/`, 2012. [Online; Stand 23. September 2012].

[11] Michael Ossmann. Ubertooth One. `http://ubertooth.sourceforge.net/hardware/one/`, 2012. [Online; Stand 23. September 2012].

[12] International Electrotechnical Commission. Radio-frequency connectors, 1979.

[13] Texas Instruments. CC2400 Datasheet. `http://www.ti.com/lit/ds/symlink/cc2400.pdf`, 2008. [Online; Stand 23. September 2012].

[14] Texas Instruments. CC2591 Datasheet. `http://www.ti.com/lit/ds/swrs070a/swrs070a.pdf`, 2008. [Online; Stand 23. September 2012].

[15] Thierry Zoller. BTcrack OSS - Source code. `http://blog.zoller.lu/2009/02/btcrack-oss-source-code.html`, 2009. [Online; Stand 23. September 2012].

doi: 10.2312/NET-2013-02-1_03