Datavetenskap

Anders Heimer
Jonas Seffel

Säkerhetsanalys av Bluetooth-kommunikation

Examensarbete C-nivå

2006:09

Säkerhetsanalys av Bluetooth-kommunikation

Anders Heimer
Jonas Seffel

Denna uppsats är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Anders Heimer

Jonas Seffel

Godkänd, 20060607

Handledare: Christer Andersson

Examinator: Stefan Lindskog

Sammanfattning

Detta examensarbete beskriver resultatet av en undersökning av säkerheten hos Bluetooth samt hur användning av Bluetooth-utrustade produkter kan påverka användarnas integritet. Undersökningen har genomförts dels genom litteraturstudier och dels genom implementation av en forskningsprototyp. Forskningsprototypen har använts vid ett experiment där vi utrönt vilken information en Bluetooth-utrustad produkt avslöjar till en tredje part. Vid experimentet har vi även försökt koppla ihop Bluetooth-utrustade produkter med fysiska personer i syfte att visa möjligheterna att kartlägga människors rörelsemönster.

Resultatet av undersökning har visat att det finns flera säkerhetsbrister hos Bluetooth-utrustade produkter, men att merparten av dessa inte beror på Bluetooth-standarden utan på implementationen av den. Vidare dras slutsatsen att det går att kartlägga människors rörelsemönster med hjälp av Bluetooth.

I uppsatsen beskrivs först Bluetooth-standarden, följt av en beskrivning av säkerheten hos Bluetooth. Slutligen beskrivs implementationen av forskningsprototypen och resultatet av experimentet.

Security Analysis of Bluetooth Communication

This report describes the result of an experiment that studies security and privacy issues in products using Bluetooth. The study has been carried out by performing a literature study and by implementing a research prototype. The research prototype has been used in an experiment in which we examined the amount of information a Bluetooth equipped product reveals when being used. We have also tried to correlate Bluetooth equipped products with their respective owners in the experiment. This was done to investigate the possibilities of tracking peoples' movement patterns.

The result of the study shows that there exist several security flaws in Bluetooth equipped products. However, the main part of the flaws are not a product of the Bluetooth standard. Instead, it is the implementations of the Bluetooth standard that contains the security flaws. Further, we concluded that it is possible to track peoples' movement patterns using Bluetooth.

The report starts with a description of the Bluetooth standard, followed by a description of various aspects regarding Bluetooth security. Finally, the implementation of the research prototype and the subsequent results from the experiment are described.

Innehåll

Figurer

Tabeller

Kapitel 1

Inledning

1.1 Bakgrund

Bluetooth är en standard för trådlös överföring av data, till exempel ljud eller video. Bluetooth använder sig av radiokommunikation på 2,4 Ghz bandet, och räckvidden uppskattas till cirka 10 meter vid normala förhållanden och maximalt 100 meter vid gynnsamma förhållanden. Det vanligaste applikationsområdet för Bluetooth är mobiltelefoner och mobiltelefontillbehör, som exempelvis hands-free. Då nästan samtliga nytillverkade mobiltelefoner och många laptop-datorer är utrustade med Bluetooth är det högintressant att ta reda på vilka säkerhetsrisker och integritetskränkningar Bluetooth medför eller riskerar att medföra.

1.2 Syfte och målsättning

Syftet med examensarbetet är att genom implementation av en forskningsprototyp utröna hur mycket information en Bluetooth-utrustad produkt lämnar ifrån sig under användning eller i standby-läge. Forskningsprototypen är en Bluetooth-sniffer som vi kommer att utveckla i ett lämpligt programspråk. När experimentet med Bluetooth-sniffern avslutats

kommer resultatet analyseras utifrån ett säkerhets och integritetsperspektiv. Vi kommer även att genom litteraturstudier undersöka kända säkerhetsbrister för Bluetooth.

1.3 Genomförande

För att lyckas med programutvecklingen har vi läst på om Bluetooth-standarden och vilka bibliotek för Bluetooth-utveckling som finns att tillgå för olika programspråk. Därefter har Bluetooth-sniffern implementerats och experimentet planerats och utförts. Vi har även genom litteraturstudier undersökt säkerhetsmekanismer och kända säkerhetsbrister i Bluetooth.

1.4 Metod

Examensarbetet bygger på att dels genom litteraturstudier och genom experiment beskriva säkerhetsbrister hos produkter som använder sig av Bluetooth. Vid litteraturstudien har relevanta forskningsartiklar och böcker studerats i syfte att kunna beskriva kända säkerhetsbrister i Bluetooth. Genom experimentet med Bluetooth-sniffern har vi utrönt vilken information som en Bluetooth-utrustad produkt släpper ifrån sig. Experimentet har genomförts på Karlstads universitet eftersom det är en plats där många personer rör sig, och de flesta av studenterna och personalen kan antas bära mobiltelefoner.

Kapitel 2

Bluetooth

Det här kapitlet börjar med att förklara varför Bluetooth skapades, hur det fick sitt namn och vilka som står bakom teknologin. Vi förklarar även hur nätverk mellan Bluetooth-enheter uppstår, följt av en beskrivning av protokollstacken. Vidare beskriver vi vad profiler är, hur de används och hur de är relaterade till Bluetooth-applikationer.

2.1 Historia

Harald Blåtand var kung av Danmark mellan 940-950 e.kr. Under hans styre förenade han Danmark och Norge samt förde kristendomen till Skandinavien [1]. Eftersom Bluetooth-teknologin kommer från Sverige tog man namnet Bluetooth (även kallat blåtand) för symbolisera att teknologin skulle sammanföra multinationella företag, precis som Harald Blåtand förenade Danmark och Norge.

Namnet Bluetooth refererar till en öppen specifikation skapad av The Bluetooth Special Interest Group (SIG) [27]. Specifikationen beskriver en teknologi för trådlös överföring av röst eller data på korta avstånd (10—100m). Bluetooth skapades av ingenjörer på Telefonaktiebolaget LM Ericsson (Ericsson). Ericsson började 1994 undersöka om det var möjligt

[1] Vissa påstår att Harald Blåtand fick sitt namn för att han ofta hade blåa tänder på grund av att han åt mycket blåbär.

Figur 2.1: Illustration av Harald Blåtand.

att ta fram en teknologi som var strömsnål och billig, något som är viktigt för små batteridrivna enheter som mobiltelefoner. Detta var främst för att man skulle kunna ersätta kablar mellan telefoner och deras tillbehör.

För att lyckas sprida teknologin så mycket som möjligt kunde den inte vara ägd av ett enda företag. Ericsson, Intel, IBM, Nokia och Toshiba gick 1998 samman för att skapa en öppen specifikation. Allt eftersom specifikationen växte ökade antalet medlemmar i SIG, där många endast är med som “adopters”; en adopter har tillgång till licensen men kan inte påverka utformningen av nya versioner av Bluetooth-standarden. De kan däremot kommentera tidiga utgåvor av den kommande versionen av specifikationen. Idag finns det över 4500 adopters [26]. I december 1999 anslöt sig även de stora aktörerna 3com, Lucent Technologies, Microsoft och Motorola till SIG. Samtliga dessa företag hade redan gjort bidrag till specifikationen som adopters. SIG har delat upp ansvaret för olika delar av specifikationen i olika grupper (för att läsa mer om grupperna, se [28]). Version 1 av specifikationen bestod av mer än 1500 sidor.

2.2 Teknologin bakom Bluetooth

2.2.1 Underliggande arkitektur i Bluetooth

Detta avsnitt introducerar några grundläggande tekniker bakom Bluetooth-kommunikation, till vilka vi kommer att referera till i kommande avsnitt. Dessa tekniker är frekvenshoppning, roller i Bluetooth och konceptet modul och värd.

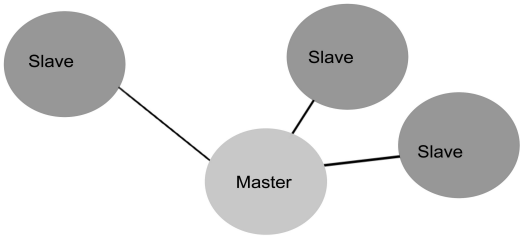
- Bluetooth använder sig av så kallad frekvenshoppning. Detta innebär att hela frekvensspektrumet (2.402–2.480 GHz) används vid Bluetooth-kommunikation. När en enhet sänder en fil kommer varje paket som sänds att ligga på en ny frekvens. Störningar från andra trådlösa teknologier hålls då på ett minimum. Förloras ett paket görs en omsänding på en ny frekvens. Enheter som kommunicerar med varandra måste därför vara synkroniserade mot en klocka för kunna följa frekvenshoppningen. Denna klocka står den enhet som kallas för "master" för. Enheterna som är synkroniserade mot mastern kallas för "slavar".

- Rollerna master och slav har en endast en betydande roll i de nedre lagren i protokollstacken (se avsnitt 2.2.3). I de övre lagren ses kommunikation som Peer-To-Peer (P2P). Vid P2P-kommunikation har de kommunicerande enheterna inga speciella privilegier gentemot övriga enheter. Slutligen kan nämnas att den traditionella klient-server-modellen från datakommunikationsvärlden används av applikationer som har implementerat vissa profiler (se avsnitt 2.2.4).

- En Bluetooth-enhet består av två delar — en modul och en värd. Modulen (exempelvis en USB-adapter) står för Bluetooth-teknologin (hårdvaran) och värden (exempelvis en dator till vilken USB-adaptern är ansluten) kommunicerar med den. Modulen innehåller radio-, basbands-, och link manager-lagret (vilka är placerade längst ned i protokollstacken). Värden står för L2CAP-lagret samt andra övre lager som kan behövas för att implementera de tjänster som ska kunna utföras. Se avsnitt 2.2.3 för

en genomgång av protokollstacken. Modulen är den faktiska hårdvaran (elektroniken) som möjliggör Bluetooth-kommunikation och värden är den enhet som som kommer utnyttja den. Ofta är modulen extra-utrustning, men så behöver det inte alltid vara, i exempelvis mobiltelefoner är modulen och värden placerade i samma enhet.

2.2.2 Bluetooth i nätverk



Figur 2.2: En piconät med en master och tre aktiva slavar.

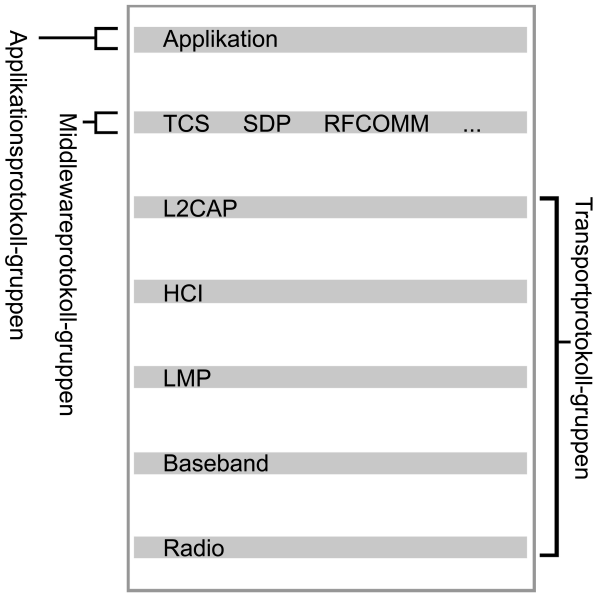
Bluetooth-kommunikation sker i så kallade ad hoc-nätverk. Detta innebär att ett nätverk av enheter kan skapas var som helst och när som helst. När enheter går samman och skapar ett ad hoc-nätverk skapas ett så kallat piconät (se Figur 2.2). I piconät finns det ingen server, utan nätet innefattar istället två roller — master och slav. Det är oftast enheten som påbörjat uppkopplingen som blir master och de kontaktade enheterna blir då slavar. En master har inga speciella privilegier, men mastern bestämmer frekvenshoppningen (se avsnitt 2.2.3). När en eller flera enheter i ett piconät ingår i ett annat piconät kallas de piconäten tillsammans för ett "scatternet". En enhet kan då vara master i ena piconätet, och slav i det andra samt viceversa.

En master i ett piconät kan kommunicera med 7 *aktiva* slavar och 255 *parkerade* slavar; de aktiva slavarna är synkroniserade med mastern och har en uppkoppling till piconätet, de parkerade slavarna är synkroniserade med mastern men har ingen annan uppkoppling till piconätet. Eftersom mastern bara kan hantera sju aktiva enheter kan en aktiv enhet bli parkerad för att kunna göra en parkerad enhet aktiv. För att en aktiv slav ska kunna

fortsätta att vara synkroniserad mot masterns klocka skickar mastern kontinuerligt paket till slaven. Dessa paket används även av slaven för att kontrollera om den kan sända data tillbaka till mastern.

För att minska batterikonsumtion kan en slav placeras i ”sniff-läge”. Då lyssnar den endast vid givna tidsintervaller efter synkroniseringspaket. Nackdelen med sniff-läge är att responsen inte kommer direkt från slaven som den gör när den är aktiv. Ett annat läge kallat ”hold” minskar batterikonsumtionen ytterligare för slaven. När slaven är i hold-läge lyssnar den inte på mastern under en given tidsperiod. Slaven kan då istället kommunicera med en annan master, eller vara master i ett annat piconät. Slavar som är satta i aktivt-läge, sniff-läge och hold-läge är alla aktiva enheter. De parkerade enheterna måste likt de aktiva enheterna vara synkroniserade med mastern.

2.2.3 Protokollstacken i Bluetooth



Figur 2.3: Protokollstacken i Bluetooth.

En protokollstack är en abstrakt beskrivning över hur data bearbetas vid datakommunikation. Protokollstackar är generellt uppbyggda av ett antal lager, där varje lager använder tjänster som ges av lagret under och tillhandahåller tjänster för lagret över. Varje lager kan innehålla ett eller flera protokoll. Protokollen i sin tur agerar som gränssnitt för olika tjänster. Lager som är placerade i botten av stacken innehåller protokoll för att kommunicera direkt med hårdvaran. De lager som är placerade i toppen tillhandahåller protokoll som möjliggör kommunikation mellan applikationer över nätverk. I tillägg till protokoll och lager finns ytterligare en abstraktionsnivå i Bluetooth — så kallade grupper. Dessa grupper innehåller ett eller flera lager.

Bluetooth-protokollstacken (se Figur 2.3) delas in i tre olika grupper vilka är hierarkiskt ordnade från hög till låg nivå.

- *Applikations-gruppen* består av de applikationer som använder sig av trådlös Bluetooth-kommunikation. Dessa applikationer kan både vara ovetande om att de använder Bluetooth (exempelvis en webbläsare), eller vara medvetna om dettta (exempelvis applikationer som använder söker efter andra Bluetooth-enheter). Rent tekniskt ingår inte applikations-gruppen i protokollstacken, utan den använder sig de tjänster som protokoll-stacken erbjuder.

- *Middlewareprotokoll-gruppen* innehåller protokoll som används för att exempelvis möjliggöra kommunikation mellan Bluetooth-enheter och befintliga applikationer som använder sig av seriell-kommunikation. Gruppen är inte bara bakåt kompatibel innehåller även protokoll som används i nyutvecklade Bluetooth-applikationer.

- *Transportprotokoll-gruppen* innehåller protokoll som hjälper enheter att hitta varandra samt att konfigurera och hantera både fysiska och logiska kommunikationslänkar. Den här gruppen befinner sig på en hårdvarunära nivå.

Anledningen till att de olika delarna delas in i grupper är för att de inte ska blandas ihop med OSI-modellen (Open System Interconnection). OSI-modellen är en mo-

dell som tagits fram av International Organization for Standardization [5]. Den definierar ett ramverk för hur nätverksprotokoll ska implementeras i sju lager. Transportprotokoll-gruppen ska därför inte blandas ihop med transportlagret i OSI-modellen, istället motsvarar transportprotokoll-gruppen fys- och länklagret i OSI-modellen.

Lager i middlewareprotokoll-gruppen

- *Radio Frequency Communication (RFCOMM) lagret*: ett av vanligaste gränssnitten för mindre enheter att kommunicera med datorer är de seriella portarna. Eftersom Bluetooth har kommit till för att ersätta kablar finns RFCOMM-lagret som tillhandahåller en abstraktion av en serieport. Detta gör det möjligt för applikationer som använder sig av seriell kommunikation att använda Bluetooth utan behöva göra ingrepp i källkoden.

- *Service Discovery Protocol-lagret*: för icke-mobila nätverk, till exempel ett nätverk av stationära datorer, finns det inget enkelt sätt att ta reda på vilka tjänster som en dator erbjuder. Säg exempelvis att du behöver skicka över filer till en central lagringsplats – hur vet du adressen till den? Kan du inte adressen utantill får du fråga systemadministratören efter adressen. Ett alternativt sätt skulle kunna vara att utföra en så kallad portscanning på nätverket, men detta anses ofta som förberedelse för dataintrång och är därför inte så lämpligt. Dessutom kräver det att användaren har kunskap i datakommunikation. Eftersom Bluetooth skapar ad hoc nätverk fanns det behov av en bättre lösning; enheten som erbjuder en lagringsplats kanske inte alltid är den samma. För att lösa detta problem används Service Discovery Protocol (SDP). SDP definierar ett sätt för enheter att beskriva tjänster som är åtkomstbara för andra enheter i nätverket.

- *IrOBEX*: eftersom Bluetooth har mycket gemensamt med infraröd trådlös överföring har SIG infört protokoll skapade av Infrared Data Association (IrDA) [1] i specifi-

kationen. Ett av de mer kända protokollen är Infrared Object Exchange (IrOBEX, oftast benämns det dock som bara OBEX). IrOBEX är ett protokoll för att skicka dataobjekt som exempelvis kalenderinformation e-post och bilder. Många av Bluetooth:s profiler (se avsnitt 2.2.4) bygger på IrOBEX.

- *Telephony Control System (TCS) lagret*: Bluetooth kom ursprungligen till för att ersätta kablar. Ren data är inte det enda som färdas i kablar, även röst gör det (i form av data). TCS-lagret är framtaget för att kunna hantera vanliga telefonifunktioner som exempelvis är kontroll av samtal eller gruppsamtal. TCS kan också användas för att initiera så kallade datasamtal. Vid datasamtal kommunicerar enheten med en dator. Ett typiskt datasamtal är användning av "dial-up networking"-profilen (se avsnitt 2.2.4) för att exempelvis koppla upp sig mot sin Internet-leverantör. I detta fall färdas data via L2CAP-lagret (se nedan) istället för en vanlig ljudkanal. Protokollet som används i TCS-lagret kallas för TCS-BIN för att det använder binär datakodning.

Lager i transportprotokoll-gruppen

- *Logical link control and adaption protocol (L2CAP) lagret* är det översta lagret i transportprotokoll-gruppen. L2CAP tar emot data från en applikation och gömmer information som är onödig för underliggande lager (exempelvis hur frekvensen hoppar eller hur paketen ser ut). L2CAP stöder "protokoll-multiplexing", vilket möjliggör att olika protokoll och applikationer delar samma luftgränssnitt (kopplingen mellan två trådlösa enheter). L2CAP tillhandahåller segmentering av stora paket som skickas av lager på en högre nivå och desegmentering av paket som kommer från en lägre nivå. Det innehåller även kontroller för att bestämma nivån på tjänster som ska ges till applikationer. L2CAP kan kontrollera inkommande trafik och samarbeta med de undre lagren för att uppehålla kvalitén på en tjänst. Detta kan exempelvis innebära att L2CAP reserverar resurser (det vill säga bandbredd) för att kunna tillhandahålla

god kvalité vid röstöverföring[2].

- *Host Controller Interface (HCI) lagret*: Bluetooth-moduler tillverkas av många olika företag. För att kunna kommunicera med olika moduler på ett standardiserat sätt används HCI, vilket är ett gränssnitt som möjliggör kommunikation mellan en modul och dess värd, exempelvis en persondator (se avsnitt 2.2.1).

- *Link Manager lagret*: länkhanterare i varje enhet bestämmer vilka egenskaper ett luftgränsnitt ska ha, bland annat mängden nödvändig bandbredd för att kunna tillgodose graden QoS som L2CAP har bestämt. Detta görs med hjälp av Link Manager Protocol (LMP). Länkhanterare övervakar också enheter när de går samman till ett par ("pairing"), se avsnitt 3.1.2. När två enheter skapar ett par skapas ett förtroende mellan enheterna genom att en autentiseringsnyckel genereras och lagras för framtida autentisering och eventuell kryptering av data (se avsnitt 3.1.3). Om autentiseringen mellan enheterna misslyckas kan respektive länkhanterare koppla ned länken mellan dem och förbjuda vidare kommunikation.

- *Basbandslagret*: Exempel på uppgifter som basbandslagret utför är att initiera ett luftgränssnitt, definiera hur enheter kan söka efter varandra, samt hur de kopplar upp sig mot varandra. Basbandslagret definierar också de tidigare introducerade rollerna master och slav (se avsnitt 2.2.1). Basbandslagret specificerar vidare hur frekvenshoppningssekvenserna ska se ut för de kommunicerande enheterna. Regler för hur ett luftgränssnitt ska delas mellan flera olika enheter baseras på en metod som kallas Time Division Duplex (TDD). Detta innebär att vid jämna tidsintervaller skickar en master data och vid udda tidsintervaller skickar en slav data. En slav synkroniserar sin klocka för att följa masterns frekvenshoppning. Basbandslagret specificerar även vilka paket som stöds vid synkron och asynkron trafik. Synkron trafik kräver efter en tid svar från mottagaren, till exempel en bekräftelse på att ett paket kommit

[2]I datakommunikation kallas detta generellt för Quality of Service (QoS) [33].

fram. Detta är inte fallet med asynkron trafik. Basbandslagret har slutligen stöd för sändningar och omsändningar, felhantering, kryptering och det kan även göra signaler oförståeliga för utomstående — likt betal-TV kanaler.

- *Radiolagret*: Radiolagret är det lager som befinner sig längst ned i stacken, och det befinner sig därmed närmast hårdvaran (se Figur 2.3). Som tidigare nämnts använder sig Bluetooth av ett frekvensspektrum mellan 2.402–2.480 GHz. Radiolagret har till uppgift att se till att enheten håller sig inom detta intervall. Vidare styr radiolagret sändningskraften hos enheten. Bluetooth delas in i tre kraftkategorier:

  – Kategori 1: klarar en räckvidd på c:a 100m, sändningskraft – 20dBm.

  – Kategori 2: klarar en räckvidd på c:a 10m, sändningskraft – 4dBm.

  – Kategori 3: klarar en räckvidd på c:a 0.1m, sändningskraft – 0dBm.

  Enheter som kan kontrollera signalstyrkan kan avläsa den andre enhetens signalstyrka (RSSI – Remote Signal Strength Indicator) via LMP. Om signalen är för svag ökas den egna signalstyrkan för att förbättra uppkopplingen mellan enheterna, är signalen istället mer än tillräcklig kan den sänkas för att spara batteriåtgång [34].

2.2.4 Profiler i Bluetooth

Istället för att skapa ett Application Programming Interface (API) för Bluetooth har SIG valt att skapa så kallade profiler. En profil beskriver ett specifikt användningsscenario för Bluetooth, samt hur protokollstacken kan användas för att implementera detta scenario. Profiler är alltså i sig inga applikationer utan de är snarare riktlinjer för hur vissa delar av applikationen ska implementeras. Likt dokumenten Request For Comments (RFC), se [6], är de nödvändiga för att uppehålla en standard så att enheter och applikationer kan kommunicera med varandra oavsett fabrikat [3]. Profilerna kan delas in i fyra olika grupper:

[3]RFC-dokument beskriver exempelvis hur olika protokoll implementeras

- *Generic*: denna grupp innehåller Generic Access Profile (GAP) och Service Discovery Application Profile (SDAP). Dessa profiler är så kallade basprofiler vilka de flesta andra profiler implementerar. GAP och SDAP var bland de sista profilerna som SIG specificerade i Bluetooth standarden. När SIG hade specificerat ett antal olika profiler såg de att de flesta av dem hade ett liknande mönster. För att underlätta för utvecklare tog SIG fram profilerna GAP och SDAP. Nu är det obligatoriskt att implementera åtminstone GAP i en Bluetooth-enhet.

  – GAP går hand i hand med transportprotokollen. Profilen beskriver hur enheter kopplar upp sig och söker efter andra enheter. GAP ansvarar för pairing och bestämmer hur länken mellan enheter ska konfigureras samt vilken säkerhetsnivå som ska användas. Säkerhetsnivån bestämmer om enheten ska kräva autentisering och eventuellt även kryptering. En enhet kan bara använda sig av en säkerhetsnivå åt gången. Den kan alltså inte använda olika säkerhetsnivåer vid kommunikation med flera olika enheter. GAP erbjuder följande säkerhetsnivåer:

    * *Nivå 1*: den här nivån erbjuder ingen säkerhet. Enheten agerar aldrig verifierare och skickar därmed aldrig så kallade *LMP_in_rand* eller *LMP_au_rand* meddelanden[4]. Detta medför att en enhet kan börja kommunicera med en enhet vilken är illvillig.
    * *Nivå 2*: Ingen mottagningen eller sändning sker innan enheten har mottagit ett så kallat *L2CAP_Connection_Request* kommando. När ett sådant kommando mottas reserveras en logisk länk mellan de kommunicerande enheternas L2CAP-lager (se avsnitt 2.2.3). Säkerhetsnivån kan ökas beroende på vilken tjänst som efterfrågas.

[4]Om en verifierare skickar ett *LMP_au_rand* meddelande (innehåller ett 16 byte stort slumpmässigt tal) till enheten som önskar att skapa en uppkoppling använder den mottagande enheten länknyckeln på det mottagna numret och sänder tillbaks resultatet. Är resultatet det förväntade anses enheten autentiserad. *LMP_in_rand* meddelanden fungerar på samma sätt, fast operationen börjar med att enheterna utför pairing för att ta fram en gemensam länknyckel.

    * *Nivå 3*: Den här nivån placerar säkerhetsskyddet i link manager-lagret (se avsnitt 2.2.3). Ingen kommunikation görs med de övre lagren innan enheten som önskar kommunicera är autentiserad i detta lager. Denna nivå placerar alltså säkerhetsskyddet i modulen istället för i värden som nivå 2 gör, och kan därmed anses säkrare.

  – SDAP går högre upp i stacken än vad GAP gör. Profilen beskriver hur applikationer som använder SDP (se avsnitt 2.2.3) för att söka efter tjänster ska implementeras. SDAP:s uppgift är att söka efter tjänster som erbjuds hos andra enheter, vilket oftast sker så fort en koppling skapas mellan två enheter. SDAP behöver bara implementeras i enheten som *söker* tjänster. Enheten som tar emot en förfrågan om en viss tjänst finns eller inte använder protokollet SDP för att svara.

- *Serial*: profilerna i den här gruppen används för att skicka och ta emot filer och kalendrar.

  – Serial Port Profile (SPP) definierar nödvändiga operationer för att använda RFCOMM-protokollet (se avsnitt 2.2.3) mellan två enheter, samt vilka protokoll som ska användas för att emulera kommunikation över en serieport.

  – Generic Object Exchange Profile (GOEP) definierar tydliga roller för enheter som använder OBEX-protokollet (se avsnitt 2.2.3). Till skillnad från många profiler som används i P2P-läge används här klient-server modellen. Innan något sänds mellan klienten och servern så måste en länknyckel vara skapad och verifierad.

  – Object Push Profile (OPP) specificerar hur data skickas från en klient till en server. Klienten kallas här för textit“push klient” och servern för textit“push server”. OPP definierar operationerna *object push*, *business card pull* och *business card exchange* (skicka, respektive, ta emot en persons kontaktuppgifter).

Object push, en funktion för att sända filer, är den enda funktion som är obligatorisk att implementera i OPP. De andra två funktionerna, vilka är utformade för att klienten ska kunna be om serverns visitkort, är varianter av *object push*. Egentligen ska profilen bara hantera data åt ett håll, men eftersom det skulle bli alldeles för komplext med att byta roller så har man inkluderat "business card exchange" i profilen.

– File Transfer Profile (FP) specificerar hur filer och kataloger kan utbytas mellan en server och en klient. FP är en något mer avancerad version av OPP eftersom den stöder generell sändning och mottagning av filer åt båda hållen. De flesta enheter som implementerar profilen klarar av att agera både klient och server. Profilen har stöd för följande operationer (från klientens synvinkel):

  * Mottagning av filer och kataloger (obligatorisk att implementera).
  * Sändning av filer och kataloger (obligatorisk att implementera).
  * Bläddra och navigera i kataloger (obligatorisk att implementera).
  * Borttagning av filer och kataloger.
  * Skapande av filer och kataloger.

  Den obligatoriska delen av hämtning av kataloger innebär inte att klienten kan hämta hela innehållet i en katalog. Det innebär istället att klienten kan hämta *beskrivningen* av katalogen, det vill säga vilka filer och andra kataloger som ligger i den valda katalogen. FP specificerar hur klienten kan hämta hela innehållet i en katalog, men denna del är inte obligatorisk att implementera. För att minimera riskerna med filöverföring kräver FP att en användare måste acceptera varje sändning och mottagning. Detta medför att användaren av enheten hela tiden är medveten om när en fil sänds eller tas emot, och på så sätt minskas risken för att bli utsatt för virus eller annan elakartad programvara (se avsnitt 3.2.6). Profilen har även stöd för autentisering och kryptering (se avsnitt 3.1), även om

det inte är obligatoriskt att implementera denna del.

- Synchronization Profile (SP) används för att synkronisera filer på en enhet med filer på en annan enhet. Även här används rollerna klient och server. Servern är oftast en mobil enhet, till exempel en telefon eller en handdator, och klienten är oftast en persondator. SP beskriver inte hur själva synkroniseringen av objekten ska gå till, istället tar klienten som hand om detta. Profilen specificerar inte heller vilka filtyper som kan synkroniseras, men det finns några obligatoriska filtyper som måste implementeras:
  - Telefonböcker
  - Kalendrar
  - Anteckningar
  - Meddelanden

  SP använder den högsta säkerheten av alla profiler i version 1.0 av specifikationen (nuvarande versionen av specifikationen är 1.2). Profilen kräver användning av autentisering och kryptering istället för endast ha interaktion med användaren som ett skydd. Dessa operationer hämtar SP från GAP och GOEP.

- *Telephony*: Profilerna i telephony-gruppen är tänkta att enbart hantera röstdata.

  - Cordless Telephony Profile (CTP) använder sig av TCS-BIN protokollet (se avsnitt 2.2.3) och har skapats för att låta Bluetooth-enheter agera som trådlösa telefoner. Profilen definierar rollerna “gateway” och “terminal”. Enheten som agerar gateway är en basstation som kan ses som servern i ett piconät. Terminalerna är de enheter som agerar som trådlösa telefoner. En gateway kan hantera sju aktiva terminaler åt gången. Terminalerna som är aktiva i piconätet har hela tiden en uppkoppling till den enhet som agerar gateway för att minimera tiden det tar att koppla samtalet till terminalen. För att undvika att personer

som inte har rätt att utnyttja gatewayen blir en del av piconätet så måste alla enheter autentiseras. Profilen understryker att alla samtal ska krypteras för att undvika tjuvlyssning[5].

- Intercom Profile (IntP) använder TCS-BIN protokollet för att låta en Bluetooth-enhet agera walkie-talkie. IntP definierar inga roller, utan de enheter som har en direktuppkoppling mellan varandra kommunicerar enligt P2P-modellen. Profilen kan tyckas onödig i och med att Bluetooth har en relativt kort räckvidd, men vid stora folksamlingar och konferenser där ljudnivån är hög kan den dock komma till användning.
- Headset Profile (HSP) beskriver hur kommunikationen mellan exempelvis en mobiltelefon och ett hands-free ska gå till. Profilen specificerar inte vilka roller de två kommunicerande enheterna (hands-freen och mobilen) ska ha utan SIG har lämnat det beslutet åt de som implementerar HSP. Detsamma gäller säkerhetsnivån.

- *Networking*:

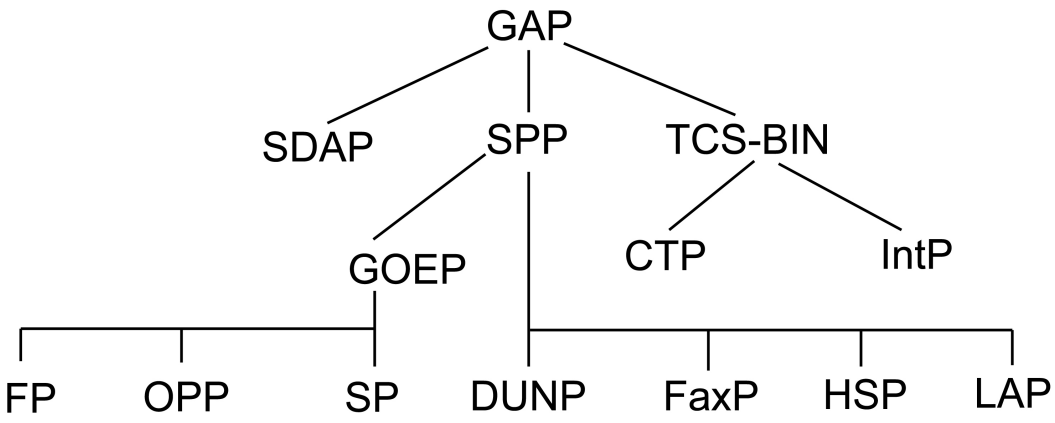
  - Dial Up Networking Profile (DUNP) tillåter en Bluetooth-enhet att agera som ett modem. Profilen baseras på SPP. DUNP definierar två roller – "gateway" och "data terminal". Enheten som är gateway agerar modem och enheten som är data terminal är den enhet (ofta en persondator) som önskar använda modemet för att komma åt till exempel Internet. Oftast använder man modem för att komma åt ett externt nätverk som Internet, men DUNP specificerar även det omvända scenariot – att modemet självt blir uppringt. För att se till att inga andra enheter kan använda sig av gatewayen så är pairing en obligatorisk del

[5]Många tidiga versioner av trådlös telefoni, till exempel barnlyssnare (möjliggör för föräldrar att lyssna på sitt barn när det sover), var utsatta för tjuvlyssning på grund av att de inte använde sig av kryptering och/eller frekvenshoppning.

att implementera. SIG rekommenderar även kryptering mellan gatewayen och data terminalen, men denna del är inte obligatoriskt att implementera.

- Fax Profile (FaxP) tillåter en persondator att skicka fax via en Bluetooth-telefon (hemtelefon eller mobil). Profilen definierar samma roller som i DUNP (gateway och data terminal). Likt DUNP kan enheten som agerar fax även ta emot data.
- LAN Access (LAP) specificerar hur en Bluetooth-enhet kan komma åt ett lokalt nätverk (LAN). LAP använder inte protokoll som finns i Bluetooth-stacken, utan profilen använder sig istället av organisationen Internet Engineering Task Force's (IETF) Point to Point Protocol (PPP). Likt DUNP är även LAP baserad på SPP. LAP definierar rollerna "access point" (accesspunkt) och "data terminal". Accesspunkten är enheten som har kontakt med det lokala nätverket, och data terminalen är den enhet som vill komma åt nätverket och innehåller även därför logiken för PPP. När en uppkoppling finns mellan accesspunkten och dataterminalen kan nätverkets olika protokoll användas mellan dem även fast de inte finns med i Bluetooth's protokollstack. Accesspunkten kan hantera flera enheter samtidigt; varje enhet har då en egen PPP-uppkoppling. Kryptering och pairing är obligatoriskt att implementera i LAP.

Figur 2.2.4 visar hur profilerna ärver funktionalitet av varandra för att undvika redundans, samt att GAP ligger till grund för samtliga profiler.



Figur 2.4: Profil hierarkin — alla profiler ärver funktionaliteten av GAP.

Kapitel 3

Säkerhet i Bluetooth

I detta kapitel diskuteras först de säkerhetsmekanismer som finns i Bluetooth: inställningar, autentisering och kryptering. Därefter diskuteras kända säkerhetsbrister i Bluetooth, som exempelvis ett antal olika sätt att ta reda på en enhets unika adress.

3.1 Säkerhetsmekanismer i Bluetooth

3.1.1 Bluetooth-inställningar

I mobiltelefoner utrustade med Bluetooth kan inställningar av telefonens synlighet och telefonnamn göras. Det går även att stänga av Bluetooth helt. När en telefon är synlig kan den upptäckas av andra Bluetooth-enheter, det vill säga den skickar sin enhetsadress till andra mobiltelefoner på begäran. När den är osynlig kan den inte hittas. För att kunna använda de profiler som är implementerade på telefonen, till exempel headset eller filöverföring (se avsnitt 2.2.4), måste Bluetooth vara påslaget och de kommunicerande enheterna känna till varandras enhetsadresser. Är telefonen osynlig måste alltså den andra enheten känna till telefonens enhetsadress i förväg för att kunna kommunicera med den. På detta sätt utgör dessa inställningar ett basalt säkerhetsskydd. Inställningsmöjligheterna finns beskrivna i Tabell 3.1.

Telefonmodell	*Synlighet*	*Bluetooth av/på*
SonyEricsson T600	Visa telefon/Dölj telefon	Av/På
Ericsson K700i	Visa telefon/Dölj telefon	Av/På
Nokia 6681	Synlig/Ej synlig	Av/På

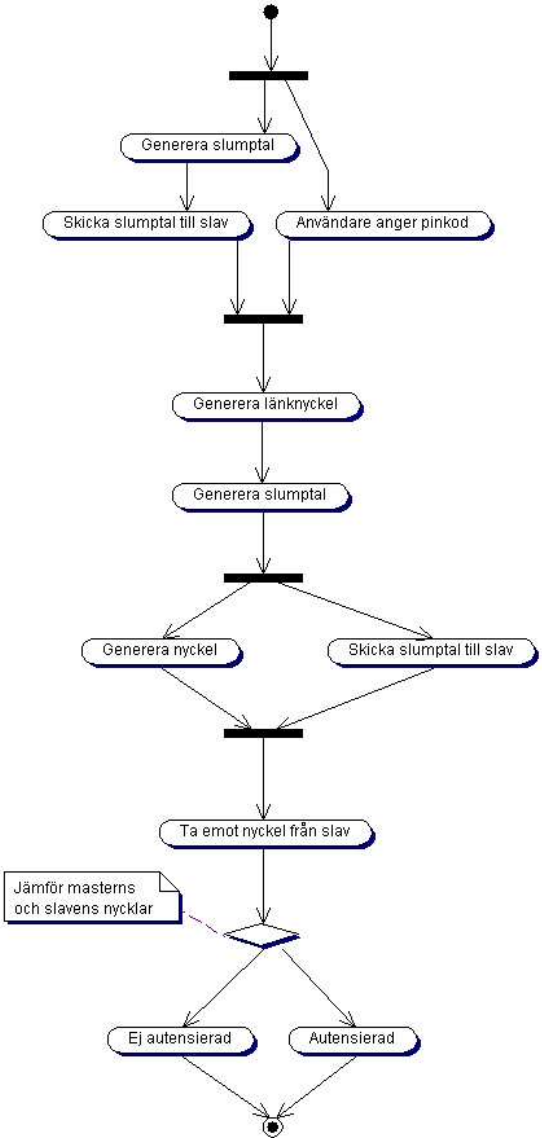
Tabell 3.1: Bluetooth-inställningar för ett antal vanliga telefonmodeller.

3.1.2 Autentisering

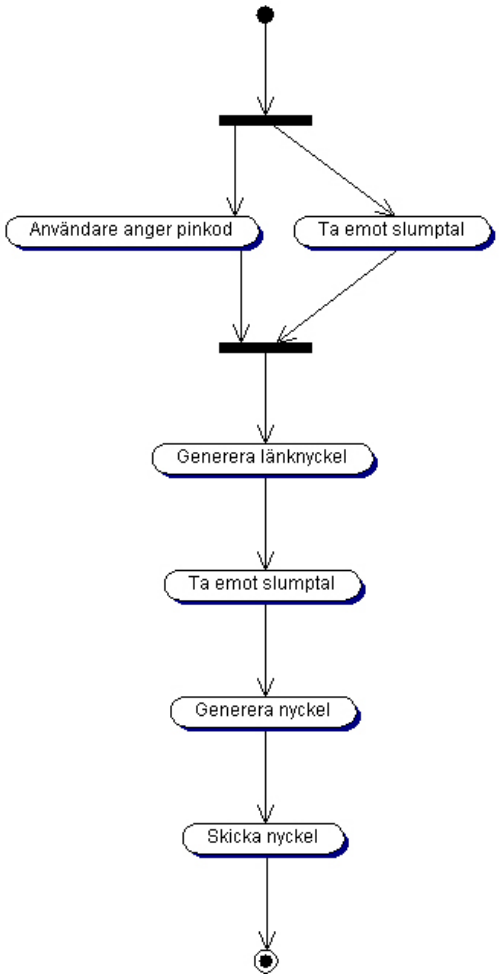
Autentisering är en process där en enhet eller person verifierar att de verkligen är den de utger sig för att vara [9]. Ett exempel på detta är när en datoranvändare verifierar sitt användarnamn med ett lösenord. Autentisering i Bluetooth går till så att den enhet som agerar master genererar ett slumptal som skickas till en slav. Därefter får både masterns och slavens användare ange en PIN-kod. Uifrån PIN-koden och slumptalet räknas det därefter ut en länknyckel hos båda parter. När länknyckeln är uträcknad sparas den och mastern skickar ett nytt slumptal till slaven. När slaven tar emot detta slumptal räknas en ny nyckel fram med hjälp av länknyckeln och slumptalet. Den nya nyckeln skickas därefter till mastern som också har räknar ut en ny nyckel på samma sätt som slaven. Om masterns nya nyckel och den nyckel som tagits emot är identiska har autentiseringen lyckats. Detta kallas i Bluetooth-terminologi för pairing. Händelseförloppet är illustrerat i Figur 3.1 och 3.2. Om autentiseringen lyckats sparas länknyckeln, vilket kallas bonding, så att användarna inte behöver ange PIN-koden på nytt. Autentiseringsprocessen börjar då med att mastern skickar det andra slumptalet.

3.1.3 Kryptering

Den krypteringsmetod som används av Bluetooth heter E_0 och är ett strömbaserad krypteringsalgoritm (varje bit krypteras var för sig). E_0 är en symmetrisk algoritm, det vill säga alla kommunicerande parter använder sig av samma hemliga nyckel. Nyckelns längd är variabel mellan 8 och 128 bitar.

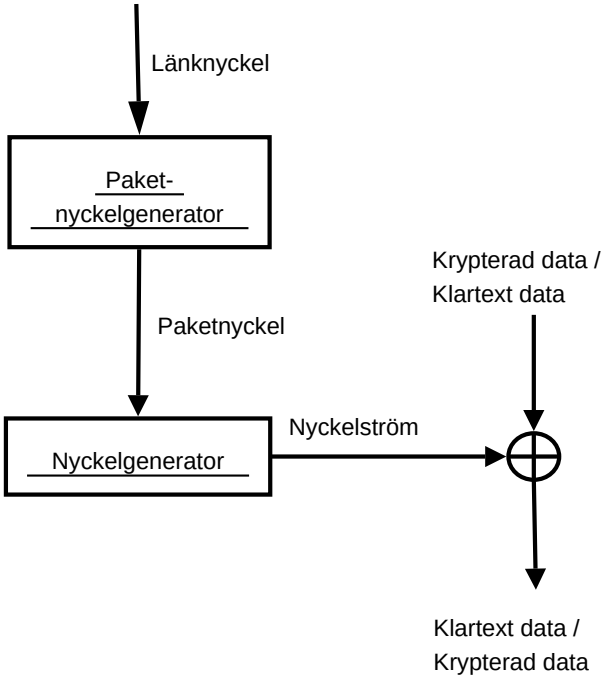


Figur 3.1: Flödesdiagram för autentisering hos mastern.



Figur 3.2: Flödesdiagram för autentisering hos slaven.

Eftersom länknyckeln för autentisering även används för att skapa krypteringsnyckeln måste autentisering ha skett mellan de enheter som vill kryptera sin trafik. Krypteringsnyckeln används sedan för att skapa en nyckelström som används när datan krypteras/dekrypteras. Processen är avbildad i Figur 3.3.



Figur 3.3: Krypterings-/dekrypteringsprocessen.

3.2 Kända säkerhetsbrister i Bluetooth

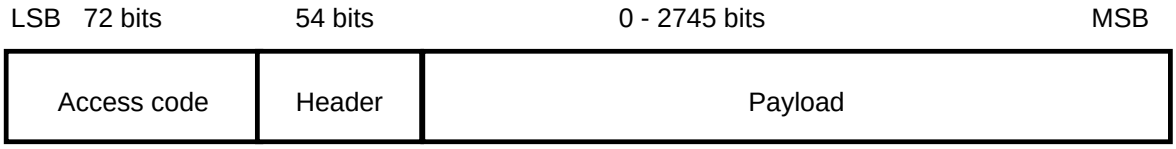
3.2.1 Avläsning av unik enhetsadress

För att kunna åtskilja Bluetooth-enheter används en 48-bitars Bluetooth-enhetsadress. Adressen, som är permanent, är unik för varje Bluetooth-enhet [22]. Exempelvis har den Bluetooth-modul vi använt vid experimentet adressen 00:0A:D9:F8:12:96.

Wong och Stajano beskriver i [35] fem olika metoder för att ta reda på en enhetsadress.

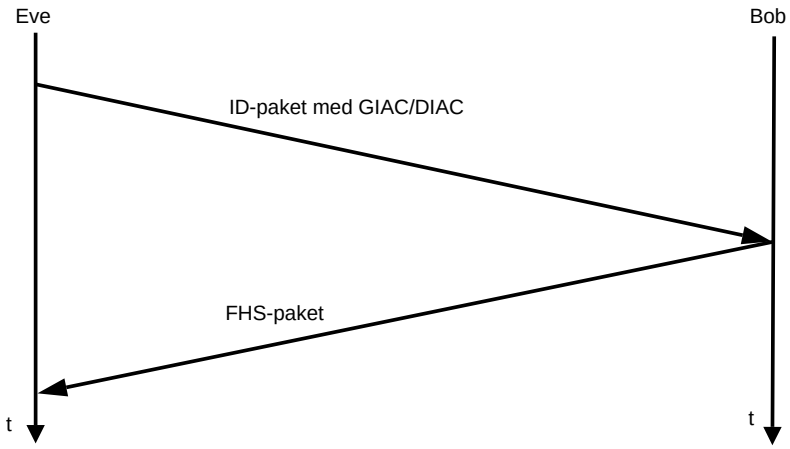
De första två sätten är så kallade *aktiva* attacker där den som attackerar måste kommunicera med offret medan de tre sista är *passiva* attacker där endast avlyssning av radiotrafik sker.

1. Det första och enklaste sättet är när Bluetooth-enheten är konfigurerad till att vara synlig. En angripare kan då med hjälp av en vanlig mobiltelefon eller persondator utrustad med en Bluetooth-modul söka efter andra Bluetooth-enheter. Sökningen går till så att den som söker efter andra enheter skickar iväg ett ID-paket med antingen en General Inquiry Access Code (GIAC) eller en Dedicated Inquiry Access Code (DIAC) i "access-code" fältet (se Figur 3.4). GIAC- och DIAC-koderna, som innehåller enhetens adress, talar om för andra enheter att de ska svara om de är synliga. De enheter som är konfigurerade till att vara synliga svarar då med att skicka tillbaka ett FHS-paket (Frequency Hop Synchronisation) med sin egen enhetsadress och tidsangivelse. Ett sekvensdiagram för hur paketen skickas finns illustrerat i Figur 3.5.



Figur 3.4: Paketstruktur hos ett ID-paket i Bluetooth.

2. Om en Bluetooth-enhet är konfigurerad till att vara osynlig ignoreras den sökande enhetens ID-paket och enheten blir därför inte upptäckt. Är den osynliga enheten konfigurerad att acceptera uppkopplingar (det vill säga Bluetooth är påslaget) kommer den dock att svara om den får ett meddelande som innehåller enhetens egna enhetsåtkomstkod (DAC). Enhetsåtkomskoden räknas fram av de sista 24-bitarna av enhetsadressen. Den som söker efter enheter kan skicka iväg paket innehållande en enhetsåtkomstkod för varje möjlig adress inom adressrymden på 24-bitar, en så kallad bruteforce-attack. Om en Bluetooth-enhet upptäcker ett sådant paket med sin egen



Figur 3.5: Sekvensdiagram för metod ett.

enhetsåtkomstkod skickar den ett svar som också den innehåller dess åtkomstkod. Om detta sker vet den som söker att det finns en Bluetooth-enhet inom räckhåll (se Figur 3.6).

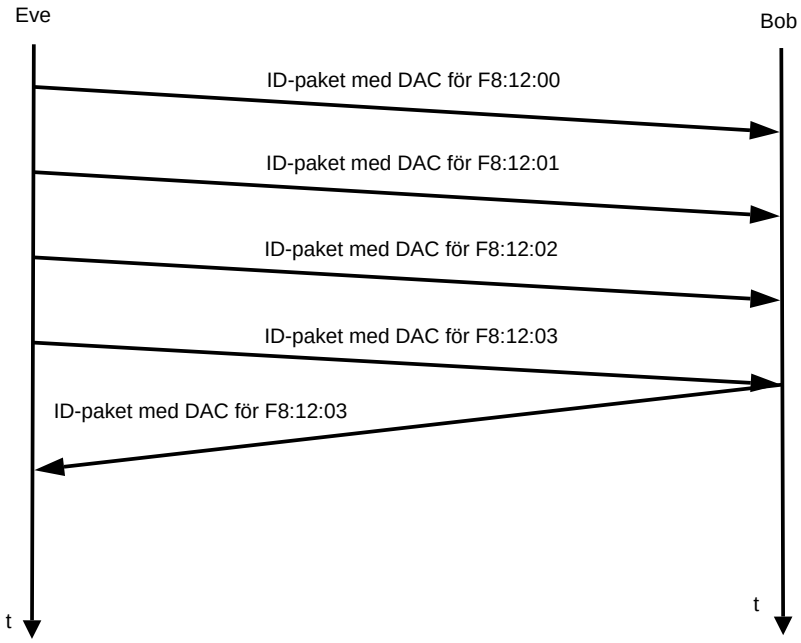
Eftersom varje tillverkare blir tilldelat en eller flera adressrymder om 24 bitar är det även möjligt att begränsa sökrymden till ett visst fabrikat av Bluetooth-utrustning [28]. Informationen om vilka adressrymder som tillhör vilka företag finns publikt tillgängligt på [3]. Den utrustning som behövs för att göra en Bruteforce-sökning är endast en Bluetooth-utrustad dator eller mobiltelefon. Om man använder 127 stycken Bluetooth-enheter, som söker av varsinna $\frac{2^{24}}{127}$ bitar av addressrymden på 2^{24} möjliga adresser, tar det elva timmar att söka igenom hela adressrymden [35]. Eftersom Bluetooth-utrustade enheter ofta är mobila är det en ganska liten chans att en Bruteforce-sökning skulle lyckas inom den tidsram en eftersökt Bluetooth-enhet är inom räckhåll.

Nedan visar pseudo-kod för hur en bruteforce-sökning som är begränsad till en av Nokias adressrymder skulle kunna gå till.

```
ADDR_SPACE nokia_start = 00:02:EE:00:00:00;
```

```
ADDR_SPACE nokia_end   = 00:02:EE:FF:FF:FF;
BD_ADDR bd_addr;

for(bd_addr = nokia_start; bd_addr <= nokia_end; bd_addr++) {
    if( -1 != get_name(bd_addr) )
        print("Found Bluetooth-device with BD_ADDR: " + bd_addr);
}
```

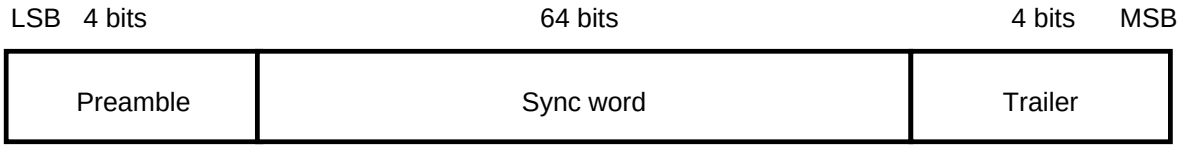


Figur 3.6: Sekvensdiagram för metod två.

3. En enhetsadress kan också bli avslöjad genom passiv avlyssning av radiotrafik. I detta fall krävs mer avancerad radioutrustning än den som vanligtvis sitter monterad i en Bluetooth-enhet. Radioutrustningen måste klara av att lyssna på flera frekvenser parallellt, eftersom Bluetooth använder sig av 79 olika radiofrekvenser. Anledningen till

att det inte går att använda en standard Bluetooth-enhet är att de är konstruerade för att använda frekvenshoppning (se avsnitt 2.2.3); en applikationsprogrammerare kan alltså inte bestämma radiofrekvensen som en viss Bluetooth-enhet skall använda. De tillfällen en Bluetooth-enhet skickar sin enhetsadress är dels vid sökning av enheter och dels vid etablering av kontakt mellan två enheter. Vid sökning av enheter sänder den sökande enheten först sin enhetsadress och när en enhet sedan svarar gör enheten det genom att skicka sin enhetsadress. När en kontakt etableras mellan två enheter börjar den som tar kontakt att skicka ett paket med den kontaktade enhetens enhetsåtkomstkod. Den kontaktade enheten svarar sedan med att skicka sin enhetsåtkomstkod. När den kontaktade tagit emot den andra enhetens svar skickar den ett paket som innehåller sin enhetsåtkomstkod som svar.

4. När en uppkoppling är etablerad mellan två enheter skickas inte längre någon enhetsåtkomstkod för enheterna med i paketen. Istället skickas en kanalåtkomstkod (Channel Access Code, CAC). I kanalåtkomstkoden finns ett 64-bitars "sync word" (se Figur 3.7) som räknas fram med hjälp av bland annat de sista 24-bitarna av enhetsadressen. Den som avlyssnar trafik mellan två Bluetooth-enheter kan göra en tabell där alla 2^{24} möjliga sync word:s och dess motsvarande enhetsadresser finns med. När en avlyssnare tar emot ett paket innehållande en kanalåtkomstkod kan avlyssnaren extrahera sync word-fältet och använda en sådan tabell för att ta reda på enhetadressen.



Figur 3.7: Strukturen hos "access-code" fältet.

5. Ett annat sätt att ta reda på en enhetsadress är när en tredje part avlyssnar kommunikationen mellan två Bluetooth-enheter och mäter hur ofta en frekvens används.

De frekvenser som en Bluetooth-enhet använder sig av, och vid vilka tidpunkter, bestäms av enhetens klocka och de sista 27-bitarna av dess enhetsadress. Om ett paket avlyssnas skulle 2^{22} enheter av 2^{28} enheter med samma klocka kunna sända på den frekvensen som avlyssnas. När nästa paket avlyssnas har även det 2^{22} möjliga avsändaradresser. Nästa steg är att de 2^{22} adresserna från första avlyssningstillfället jämförs med de 2^{22} adresserna från det andra avlyssningstillfället. När resultaten kombineras och de adresser som inte förkommer i båda delmängderna tas bort minskas antalet möjliga avsändaradresser till 2^{16}. Om sex paket totalt avlyssnas och de 2^{22} möjliga avsändaradresserna för varje paket jämförs får man fram en unik enhetsadress som med mer än 99% sannolikhet är den rätta.

Tabell 3.2 visar vilka kriterier Bluetooth-enheten måste uppfyllda för att respektive metod ska fungera. De metoder vi kommer att implementera i detta examensarbete är metod ett och metod två.

	Synlig	*Kontaktbar*	*Kommunicerande*
Metod 1	X		
Metod 2		X	
Metod 3			X
Metod 4			X
Metod 5			X

Tabell 3.2: Vilkor för avläsning av enhetsadress.

3.2.2 Bluejacking

Bluejacking är ett sätt att skicka meddelanden med hjälp av OBEX-protokollet (se avsnitt 2.2.3) och beskrivs av Potter i [25]. När en Bluetooth-enhet vill skicka ett objekt måste den först göra en förfrågan till den mottagande enheten om den vill ta emot objektet. I förfrågan skickas en namnsträng som anger vilken telefon eller person förfrågan kommer i från. Strängen, som maximalt kan innehålla 248 tecken, kan ha godtyckligt innehåll.

När förfrågan kommer till den mottagande telefonen visas namnsträngen på dess display enligt följande (beroende på telefonmodell): "Vill du ta emot en telefonbok/bild från *namnsträng*". Innehållet i namnsträngen kan till exempel användas för reklam, en resturang kan skicka "Välkommen in till Bob´s Inn och ät för 49:-" till alla förbipasserande personer med Bluetooth-utrustade mobiltelefoner. Ett annat exempel på innehåll i namnsträngen kan vara personliga meddelanden av typen "Ska vi ta en öl tillsammans?". Attacken är alltså en typ av spam för mobiltelefoner.

3.2.3 Bluesnarfing

I [24] skriver Potter att vissa mobiltelefoner kan vara åtkomliga för en så kallad Bluesnarf-attack. Ordet "snarf" är ett slanguttryck för att ladda ner en stor fil och använda den utan upphovsmannens tillstånd [32], och syftar i det här sammanhanget på att stjäla informationen från en mobiltelefon med Bluetooth-stöd. Den information en mobiltelefon kan släppa ifrån sig vid en Bluesnarf-attack är, beroende på telefonmodell, till exempel telefonboken, kalendern och IMEI-numret, där IMEI står för International Mobile Subscriber Identity. IMEI-numret är en 15-siffrig kod som identifierar telefonen mot telefonoperatören och kan användas för att "klona" telefoner. Med klona menas att mobiltelefons riktiga IMEI-nummer byts ut mot ett IMEI-nummer från en annan mobiltelefon. På så vis visar mobiltelefonen inte sitt rätta IMEI-nummer gentemot telefonoperatören, vilket kan utnyttjas av kriminella personer som vill undvika att bli positionerade på grund av sin mobiltelefon [13].

Det som gör vissa mobiltelefoner sårbara för Bluesnarf-attacker är att dess OBEX-implementation inte kräver att autentisering av den uppkopplande telefonen görs vid utbyte av elektroniska visitkort. När telefonerna är uppkopplade mot varandra skickas vanligtvis *object push*-kommandot (se avsnitt 2.2.4) för att föra över det elektroniska visitkortet (innehåller digital kontaktinformation). Istället för *object push*-kommandot används ett *get*-kommando vid en Bluesnarf-attack. OBEX *get*-kommando fungerar tvärtemot *object*

push-kommandot, det vill säga istället för att skicka ett objekt så tar det emot ett objekt, till exempel "GET phonebook.vcs". Herfurt [11] gjorde ett experiment på CeBIT mässan 2004 där han med hjälp av en persondator utrustad med två Bluetooth-enheter undersökte om han kunde läsa telefonboken från mässdeltagarnas mobiltelefoner. Herfurt hade sin utrustning, en persondator utrustad med två Bluetooth-nätverkskort, placerad i närheten av toaletterna där de flesta mässdeltagarna passerar. Av de 179 stycken Nokia 6310/6310i telefoner som var synliga lyckades Herfurt läsa telefonboken från 25% av dessa. För telefonen SonyEricsson T610 kunde 6% av telefonböckerna läsas. Herfurt skriver att resultatet påverkats av att en Snarf-attack tar cirka 30 sekunder att utföra och att mässdeltagarna förmodligen inte alltid var inom räckhåll under hela de 30 sekunderna.

3.2.4 "DoS"-attacker

"Denial-Of-Service"-attacker (DoS) är attacker där en behörig användare hindras att använda en resurs, till exempel en mobiltelefon eller en persondator [9]. I [30] beskriver Stajano och Anderson två enkla DoS-attacker avsedda för mobiltelefoner. Den första attacken gör Bluetooth-kommunikation omöjlig genom att en stark radiosignal sänds på de frekvenser som används av Bluetooth-enheter. De svaga Bluetooth-sändarna kommer då bli "överröstade" av den starkare signalen. Det beror på att radiomottagaren i Bluetooth-enheten klipper bort den svagare signalen och endast den starkare tas emot. Är signalerna jämnstarka kan mottagaren inte urskilja någon av signalerna och endast störningar tas emot. Den andra attacken är en utmattnings-attack av batteriet hos en Bluetooth-enhet. Genom att kommunicera med en Bluetooth-enhet ofta och under en lång tidsperiod kommer dess batteri att ta slut fortare eftersom strömförbrukningen ökar vid Bluetooth-kommunikation.

En annan attack är baserat på "echo request"-paket i L2CAP-lagret (se avsnitt 2.2.3). Echo request-paketen, som är av variabel storlek, används för att kontrollera upkopplingen och mäta svarstider. Dessa paket kan jämföras med så kallade ping-paket i TCP kom-

munikation [10]. Om för stora "echo request"-paket skickas kan vissa Bluetooth-utrustade enheter tillfälligt sluta fungera [10]. Denna attack är enkel att utföra då det enda som behövs är en dator utrustad med Bluetooth-nätverkskort och programvaran "l2ping", som följer med BlueZ [2].

Enligt Gianluigi [19] beror sårbarheten för DoS-attacker på så kallade buffer-overflows. Det är enligt Gianluigi brister i design och implementation av Bluetooth-stackar och Bluetooth-applikationer som för med sig att buffer-overflows kan tillåtas uppstå. En buffer-overflow inträffar när data kopieras och käll-datan är större än det allokerade minnesutrymmet där käll-datan ska placeras och storleken på käll-datan inte kontrolleras mot storleken på det allokerade minnet. En annan konsekvens av buffer-overflows kan vara att en returadress skrivs över vilket medför att otillåten kod kan tillåtas exekvera. Nedan visas exempelkod som skulle ge en buffer-overflow.

```
char source[40];
char target[20];
strncpy(target, source, sizeof(source));
```

3.2.5 Avlyssning

Om okrypterad Bluetooth-kommunikation används finns det inget som förhindrar att en angripare avlyssnar trafiken och vidare att den som avlyssnar förstår innehållet i kommunikationen [8]. Om trafiken däremot är krypterad ställs den som avlyssnar inför problemet att dekryptera trafiken. Det finns emellertid inget sätt att räkna fram nyckeln inom rimlig tid enligt Gehrmann [8]. Den tid det tar att att hitta nyckeln till data som är krypterat med E_0 (se avsnitt 3.1.3) beror på antalet kända bitar av krypterad data. I [4] skriver Fluhrer och Lucks att om man har tillgång till 2^7 bitar krypterad data är tidsåtgången för att ta fram nyckeln $O(2^{84})$ tidsenheter. Om man känner till 2^{43} bitar minskas tidsåtgången till $O(2^{73})$. Om en dator kan exekvera 2^{20} instruktioner per sekund (cirka en miljon) skulle det ta drygt 285 miljoner år att räkna fram nyckeln.

3.2.6 Virus

I [7] beskriver Furnell ett virus som använder sig av Bluetooth för att sprida sig. Viruset, vars namn är "Cabir", utgör i sig inget större hot mot telefonen ,men visar att det är möjligt för ett virus att spridas med hjälp av Bluetooth. För att kunna bli smittad måste mobiltelefonen vara synlig; när måltelefonen är inom räckhåll (normalt 10 till 100 meter) får användaren av måltelefonen en fråga om han eller hon vill ta emot en fil. Om användaren accepterar att ta emot filen får han eller hon en ytterligare fråga om filen tillåts exekvera. Svarar användaren "ja" på den frågan blir telefonen smittad och texten "Cabir" visas i displayen. Mobiltelefonen kommer sedan att ständigt, utan ägarens vetskap, söka efter ytterligare mobiltelefoner att sprida viruset till. Detta medför, eftersom strömkonsumtionen ökar när Bluetooth söker efter enheter, att batteriet tar slut fortare. Detta är alltså en slags DoS-attack. Ytterligare exempel på virus som använder sig av Bluetooth för att sprida sig är "Lasco", "Mabir" och "Pbstealer" [31]. Lasco och Mabir är baserat på Cabir, men Mabir sprider sig även via MMS och Lasco smittar även SIS-filer (SIS är ett filformat för exekverbara filer hos mobiltelefonoperativsystemet Symbian). Pbstealer stjäl telefonboken och skickar sedan iväg den med hjälp av Bluetooth. Enligt antivirusföretaget Symantec [31] uppgår antalet smittade enheter än så länge till 0 – 49 per virus, vilket är väldigt lite jämförst med persondatorer uppkopplade mot Internet. Virusens geografiska spridning betäcknas än så länge av Symantec som liten.

3.3 Säkerhetsanalys av Bluetooth

Gemensamt för Bluesnarf- och DoS-attackerna är att de uppkommit på grund av felaktigheter eller brister i implementationer av antingen Bluetooth-stackar eller applikationer som använder Bluetooth. Mjukvarutillverkarna har dock korrigerat felen i nya produkter och är numera medvetna om riskerna. Vad det gäller spridningen av virus har den ännu varit liten. Det beror troligtvis på att en Bluetooth-enhet måste vara nära en annan Bluetooth-enhet

under en ganska lång tidsperiod för att kunna sprida virus. Det faktum att användaren för tillfället måste acceptera att ta emot en fil som begränsar också spridningen av virus.

Att Bluetooth-enhetsadresserna är permanenta, och de fem metoderna att ta reda på dem, gör att personer som bär på Bluetooth-enheter kan få sitt rörelsemönster och aktuell position avslöjad. Eftersom enhetsadressen endast är en samling siffor är det intressant för en eventuell angripare att koppla ihop den med en fysik person, vilket kan göras på flera sätt. Jacobsson och Wetzel skriver i [15] att personifiering av en enhetsadress kan göras till exempel när en person har sin mobiltelefon med Bluetooth-stöd med sig när han eller hon använder sitt kreditkort. Angriparen kan då ta reda på vilka personer som gjort kreditkortsköp när en specifik Bluetooth-enhet varit närvarande och på så vis länka enhetsadressen till en person. Angriparen måste då få namn eller kreditkortsnummer från affären. En annan möjlighet för angriparen är att följa med Bluetooth-enheten till ägarens hem och därigenom länka ihop enhetsadressen med en person. Ett ytterligare sätt att koppla ihop en enhetsadress med en fysisk person är genom att utföra en Bluesnarf-attack. Vid Bluesnarf-attacken kan en mobiltelefon lämna ifrån sig IMEI-numret, telefonböcker och annan information som gör det lätt för en angripare att koppla ihop en person med en Bluetooth-enhetsadress.

Kapitel 4

Implementation

4.1 Utvecklingsmiljöer för Bluetooth

Eftersom en del i vår uppgift bestod av att implementera en forskningsprototyp var vi tvungna att välja en lämplig plattform och ett lämpligt programmeringsspråk. De programmeringsspråk vi undersökt är Java, C# och C.

I syfte att dra nytta programspråket Javas egenskaper (objektorientering, dynamisk minneshantering, etc.) har det utvecklats ett Java Bluetooth-API kallat JABWT (Java APIs for Bluetooth Programming) [17]. JABWT är standardiserat av den organisation som sköter utveckling av Java, Java Community Process (JCP), och har av JCP fått namnet JSR-82. JSR-82 är ett gränssnitt mot Bluetooth:s protokollstack som förenklar utvecklingen av Bluetooth-applikationer. Det går därför att skriva ett Java program för Bluetooth-kommunikation som fungerar på flera plattformar utan modifikation av koden. Exempel på plattformar kan vara allt från persondatorer med operativsystemet Windows XP till mobiltelefoner med operativsystemet Symbian [18]. Hade Bluetooth-applikation kommunicerat direkt med Bluetooth-stacken hade programmen varit tvungna att anpassas för varje miljö, eftersom Bluetooth-stacken skiljer sig åt miljöerna emellan. Implementationer av JSR-82 har gjorts av flera företag och finns att tillgå för till exempel PalmOS,

Linux och Windows XP [12]. En nackdel med JSR-82 är att JSR-82 inte stödjer mätning av signalstyrka [12].

Om utveckling av Bluetooth-applikationer ska implementeras för Linux miljö, finns det förutom JSR-82 enligt Kammer et al. [16] minst tre andra Bluetooth-stackar att välja på: OpenBT [29], IBM BlueDrekar [14] och BlueZ [2]. BlueZ [2], som idag är den Bluetooth-stack som är standard för Linux, är den Bluetooth-stack vi undersökt. Det beror på att IBM BlueDrekar kostar pengar att använda och OpenBT inte längre underhålls. BlueZ erbjuder åtkomst till stacken genom ett API för programspråket C. Tyvärr är BlueZ:s dokumentation obefintlig. Dock är det mesta i Bluetooth-standarden implementerat och stöd för OBEX (se avsnitt 2.2.3) kan fås med hjälp av av OpenOBEX [23].

4.2 Motivation av utvecklingsmiljö

För att kunna välja utvecklingsmiljö utvecklade vi i experimentsyfte en enkel version av en Bluetooth-sniffer för att hitta synliga Bluetooth-enheter. Vi implementerade denna sniffer i vart och ett av de tidigare nämnda programspråken C, C# och Java. Det var förhållandevis lätt att implementera sniffern i alla tre programspråken. När vi gick vidare med att implementera en identitets-sniffer som även skulle kunna hitta osynliga Bluetooth-enheter med Bruteforce-sökning visade det sig att vi inte lyckades med detta i Java eftersom JSR-82 API:et inte innehöll den funktionalitet vi behövde. Ett exmepel på en detalj som saknades var exempelvis funktionalitet för att kontakta Bluetooth-enheter genom att ange en specifik enhetsadress. Då vi undersökte möjligheterna för att implementera Bruteforce-sökningen i C# visade sig även detta vara omöjligt. Det enda alternativ som då återstod var att skriva programmet i C med hjälp av BlueZ. BlueZ visade sig ha den funktionalitet som vi behövde för att kunna söka efter både synliga och osynliga Bluetooth-enheter.

4.3 Detaljerad beskrivning av Bluetooth-sniffern

Den Bluetooth-sniffer vi har implementerat klarar av att söka efter både synliga och osynliga enheter (se avsnitt 3.2.1). När man söker efter de synliga enheter söker programmet efter enheter som är inom dess område. Den information som kan fås fram är de påträffade enheternas unika Bluetooth-adress, enheternas namn samt vad de är för typ av enhet (exempelvis mobil, dator, eller en så kallad smartphone).

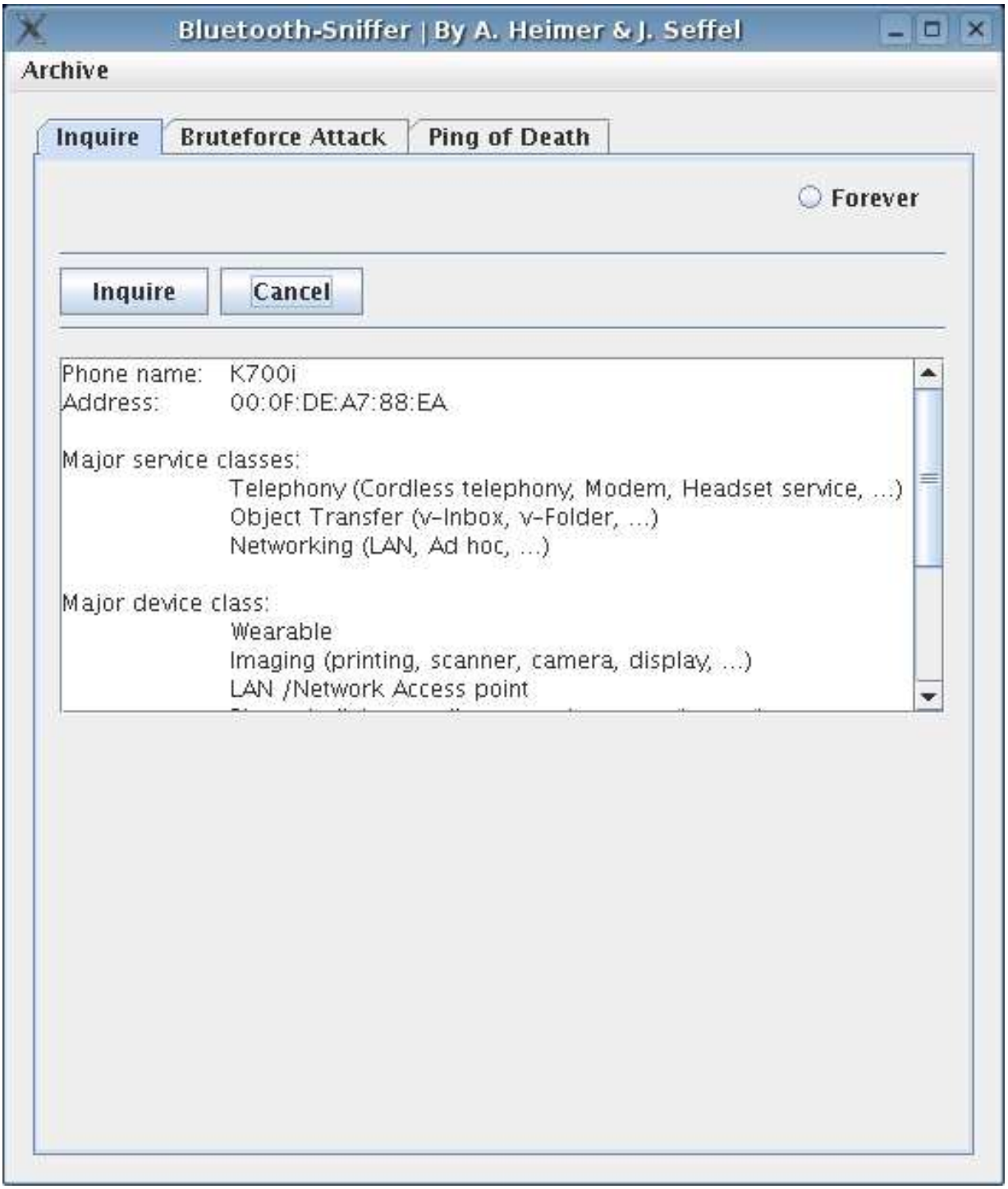
När sökning efter osynliga enheter görs användas en bruteforce-attack (se avsnitt 3.2.1). Eftersom det finns 2^{48} möjliga adresser så kan denna operation ta **lång** tid. Man kan dock välja att avgränsa sin sökning till endast Bluetooth-enheter från Nokia, SonyEricsson, eller Siemens. Adressrymden blir då 2^{24}. Informationen som fås av enhet vid en bruteforce-attack är samma som vid en sökning efter synliga enheter samt version av LMP och eventuellt även RSSI värdet (se avsnitt 2.2.3).

4.3.1 Det grafiska gränssnittet

Det grafiska gränssnittet är skrivet i Java [21], men det använder sig av funktioner som kommunicerar med Bluetooth-stacken, vilka är implementerade i C. För att kunna anropa dessa funktioner från Java har en teknik som kallas Java Native Interface (JNI) använts [20]. Vi valde att implementera gränssnittet i Java dels för att lära oss JNI och dels för att Java är ett tacksamt språk för implementation av grafiska gränssnitt.

Figur 4.3.1 visar funktionaliteten för sniffning efter synliga enheter. I detta exempel har en sökning efter närliggande enheter gjorts och en SonyEricsson K700i har påträffats. Programmet visar enhetens Bluetooth-adress, dess namn, och vad den är klassad som för typ av enhet. Väljer användaren att markera alternativet “Forever” kommer programmet att ligga och söka efter enheter kontinuerligt tills användaren trycker på “Cancel”. “Ping of Death” fliken hann vi tyvärr inte med och visas eller förklaras därmed inte.

I Figur 4.3.1 visas Bruteforce-tabben för sökning efter osynliga enheter, där användaren



Figur 4.1: Sniffer-tabben i Bluetooth-Sniffer.

Figur 4.2: Bruteforce-tabben i Bluetooth-sniffer.

själv kan specificera vilket adress-intervall denne vill söka av, alternativt välja ett redan specificerat intervall. De intervall som finns att välja mellan är de som har blivit tilldelade till Sony Ericsson, Nokia och Siemens. Denna del av programmet används främst då sökning efter osynliga enheter görs, även om synliga enheter också kommer att hittas. Tycker användaren att det tar för lång tid kan denna avbryta genom att trycka på "Cancel".

4.4 Arkitektur

Programmet består av fem delar, där vi har angett aktuell källkodsfil inom parantes:

- Det grafiska gränssnittet (`BTGui.java`): Denna del presenterar funktionaliteten för användaren.

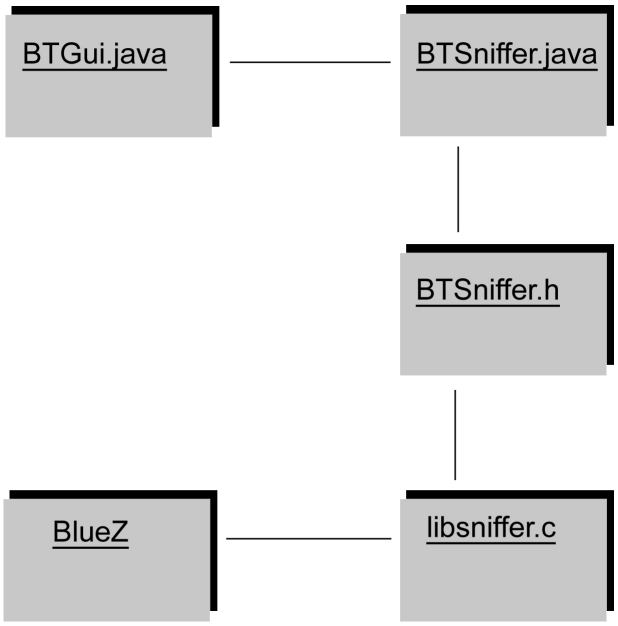
- Kopplingen mellan Java och C (`BTSniffer.java`): Innehåller funktioner för anrop till C-programmet. `BTSniffer.java` utför ingen logik själv utan skickar anropen vidare till C-programmet (`libsniffer.c`).

- Lagring av data (`Phone.java` och `PhoneData.java`): En instans av `Phone` skickas med till C-programmet när en när sökning efter synliga enheter görs, samt när använder väljer att utföra en bruteforce-attack. `Phone.java` innehåller funktioner för att lagra data i `PhoneData` klassen. När en ny enhet har hittats känner det grafiska gränssnittet av det och presenterar information för användaren.

- Biblioteket (`libsniffer.c`): innehåller logiken för Bluetooth-kommunikationen i programmet. Biblioteket har till uppgift att utföra de uppgifter som användaren önskat utföra — en sökning efter synliga enheter eller en Bruteforce-attack.

- JNI header-fil (`BTSniffer.h`): denna fil genereras av Java för att få korrekt deklarerade funktioner.

- Externa bibliotek (BlueZ, se [2]): tillhandahåller en Bluetooth-stack vilken är nödvändig för att kontakta USB-adaptern.



Figur 4.3: Bluetooth-Snifferns arkitektur.

Kapitel 5

Resultat

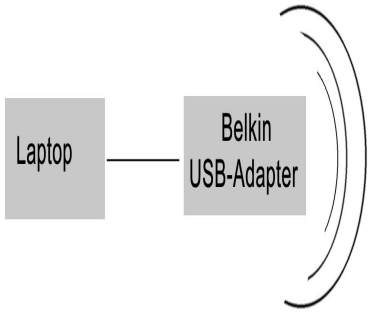
Kapitlet beskriver hur vår experimentmiljö såg ut, hur experimentet utfördes, samt slutligen resultatet av det.

5.1 Experimentmiljö

Den utrustning som användes i experimentet var en latop som vi kopplade en Bluetooth-adapter till. Laptopen som användes var en Toshiba Satellite med en Pentium 4 CPU med 2GHz klockfrekvens och 512MB internminne. Operativsystemet var Ubuntu Linux 5.10. Bluetooth-adaptern som vi använde var en Belkin F8T008. En enklare illustration av experimentmiljön kan ses i Figur 5.1.

5.2 Experimentets uppbyggnad

Experimentet genomfördes genom att vi tog med en laptop som var utrustad med en Bluetooth-adapter (se ovan) till ett av Karlstad universitet's caféer. Experimentet utfördes genom att vi lät Bluetooth-sniffern söka efter synliga enheter i 3 timmar och 15 minuter.



Figur 5.1: Illustration av experimentmiljön.

5.3 Resultat av experimentet

De enheter som påträffades när experimentet utfördes visas i tabell 5.1 [1].

Adress	*Namn*	*Antal träffar*	*Kommentar*
00:0A:D9:XX:XX:XX	Okänd	1	
00:12:EE:XX:XX:XX	Okänd	1	
00:0F:DE:XX:XX:XX	Okänd	1	
00:05:C9:XX:XX:XX	Okänd	2	
00:15:B9:XX:XX:XX	Lena	1	Närvarande i 2min.
00:60:57:XX:XX:XX	K700i	3	Närvarande i 2 min som längst.
00:16:20:XX:XX:XX	K700i	1	Närvarande i 8 min.
00:12:EE:XX:XX:XX	Z520i	7	Närvarande i 3 min som längst.
00:16:4E:XX:XX:XX	Nokia N70	6	Närvarande i 10 min som längst.
00:0E:07:XX:XX:XX	T610	2	Närvarande i 4 min som längst.
00:12:EE:XX:XX:XX	(the One N Only) 77	3	Närvarande i 10 min som längst.
00:16:20:XX:XX:XX	W800i	3	
00:14:9A:XX:XX:XX	Motorola A1000	3	Närvarande i 2min som längst
00:12:EE:XX:XX:XX	K700i	1	Närvarande i 6min

Tabell 5.1: Enheter som hittades under vårt experiment.

Tabellen visar enhetens unika Bluetooth-adress, enhetens namn samt hur många gånger enheten påträffades. Var 20:e sekund utförde Bluetooth-Sniffern en sökning efter närliggande

[1] Adresserna har censurerats för att skydda den personliga integriteten.

enheter, på så sätt kunde vi se vilka enheter som stannade kvar på platsen. Experimentet påbörjades klockan 11:15 och avslutades 14:30 den 24:e maj år 2006. De flesta enheterna påträffades under lunchtid. Efter klockan 13:00 började föreläsningarna igen för de flesta studenter, och det var då vi började försöka koppla enheter till personer. Vi lyckades identifiera åtminstone en person. Utförs experimentet under en längre period tror vi att vi skulle lyckas med att identifiera flera personer och kunna utröna deras identitet samt deras rörelsemönster.

Vid sökning efter icke synliga enheter påträffades inga enheter. Det berodde på att mobiltelefonerna förflyttar sig och en bruteforce attack tar lång tid. Det skulle ta elva timmar om vi haft tillgång till 127 stycken Bluetooth-moduler (se avsnitt 3.2.1) men vi hade bara endast tillgång till en modul. Även om vi skulle haft tillgång till så många moduler skulle vi förmodligen ändå ha misslyckats iochmed att en person sällan står på samma plats i elva timmar.

Personerna som undersöktes var inte medvetna om att experimentet utfördes. Om de underättats om detta hade vårt resultat med stor sannolikhet sett annorlunda ut.

Kapitel 6

Slutsats

6.1 Resultat

Resultatet av vår undersökning visar att Bluetooth är en standard som är förhållandevis säker. Detta beror på att Bluetooth-standarden har funnits sedan år 1998 och tillverkare som implementerar Bluetooth-produkter har lärt sig av tidigare misstag. De flesta säkerhetsbrister vi beskrivit har visat sig vara ett resultat av brister i implementationen av Bluetooth-standarden och alltså inte på själva Bluetooth-standarden. Det faktum att det går att spåra personer på grund av att de använder Bluetooth-produkter kvarstår dock. Detta beror på att Bluetooth är uppbygt med fasta enhetsadresser och på den teknik som används vid kommunikation mellan Bluetooth-enheter. Användare av Bluetooth-utrustade mobiltelefoner verkar dock ha Bluetooth avslaget eller satt till osynlig i de flesta fall, annars borde fler telefoner hittats i vårt experiment. En bidragande orsak till detta är att tillverkare har börjat leverera sina mobiltelefoner förinställda på att Bluetooth ska vara avslaget vilket förhindrar oavsiktlig (och omedveten) exponering. Den metod som vi implementerat som använder sig av en Bruteforce sökningen tar för lång tid att genomföra i praktiken och borde inte i nuläget utgöra något hot för användarna.

6.2 Framtida arbete

De attacker som hade varit intressanta att implementera i ett framtida arbete vore Bluesnarfing, Bluejacking och Denial of Service (se avsnitt 3.2.2, 3.2.3 och 3.2.4). Det hade även varit intressant att testa de passiva metoder (genom avlyssning) att ta reda på en adressen hos en Bluetooth-enhet. Vi har tyvärr varit begränsade till att endast använda en USB Bluetooth-modul vilken inte tillåter att vi kan bestämma en specifik frekvens eller frekvenser att lyssna på (se avsnitt 3.2.1).

6.3 Problem och lärdomar

De problem som uppstått under examensarbetets gång har främst varit relaterade till implementationen. Det berodde på att dokumentationen för de Bluetooth-API:n som är fritt tillgängliga är antingen undermåliga eller obefintliga, samt att vissa bibliotek inte längre underhålls alls.

Vi har lärt oss mycket om en ny protokollstack, dess likheter och olikheter med TCP/IP. En ytterligare sak vi fått större kunskap om är datasäkerhet, speciellt vid trådlös datakommunikation. Vidare har vi lärt oss att om programmeringen påbörjas innan man har undersökt ett visst bibliotek eller problem noggrannt medför detta ofta onödigt arbete. Detta är något som vi kommer att ta lärdom av i framtida arbeten och studier.

Referenser

[1] Infrared Data Association. Welcome to IrDA. Se `http://www.irda.org/`.

[2] Bluez. Bluez - Official Linux Bluetooth protocol stack. Se `http://www.bluez.org/`.

[3] IEEE Standards Department. IEEE OUI and Company_id Assignments. Se `http://standards.ieee.org/regauth/oui/index.shtml`.

[4] Scott Fluhrer and Stefan Lucks. Analysis of the E0 Encryption System. In *Revised Papers from the 8th Annual International Workshop on Selected Areas in Cryptography*, volume 2259, pages 38–48. Springer-Verlag, January 2001.

[5] International Organization for Standardization. ISO - International Organization for Standardization. Se `http://www.iso.org/`.

[6] The Internet Engineering Task Force. Ietf rfc page. Se `http://www.ietf.org/rfc.html`.

[7] Steven Furnell. Handheld hazards: The rise of malware on mobile devices. *Computer Fraud & Security*, 5:4–8, 2005.

[8] Christian Gehrmann, Joakim Persson, and Ben Smeets. *Bluetooth Security*. Artech House, Incorporated, 1st edition, 2004.

[9] Dieter Gollman. *Computer Security*. John Wiley Sons Ltd, 1st edition, 1999.

[10] Trifinite Group. The home of the trifinite.group. Se `http://trifinite.org`.

[11] Martin Herfurt. Bluesnarfing @ Cebit 2004, 2004. Se `http://trifinite.org/Downloads/BlueSnarf_CeBIT2004.pdf`.

[12] Bruce Hopkins and Antony Ranjith. *Bluetooth For Java*. Springer-Verlag, 1st edition, 2003.

[13] Jukka Hynninen. Experiences in Mobile Phone fraud. In *Proceedings of Seminar on Network Security*, volume Tik-110.501. Helsinki University of Technology, fall 2000.

[14] IBM. alphaworks: Bluedrekar: Overview. Se `http://www.alphaworks.ibm.com/tech/BlueDrekar/`.

[15] Markus Jakobsson and Susanne Wetzel. Security Weaknesses in Bluetooth. In *CT-RSA 2001: Proceedings of the 2001 Conference on Topics in Cryptology*, pages 176–191, London, UK, 2001. Springer-Verlag. LNCS 2020.

[16] David Kammer, Gordon McNutt, Brian Senese, and Jennifer Bray. *Bluetooth Application Developer´s Guide: the Short Range Interconnect Solution*. Syngress Publishing, Inc., 1st edition, 2002.

[17] C Bala Kumar, Paul J. Kline, and Timothy J. Thompson. *Bluetooth Application Programming With the JAVA API*. Morgan Kaufmann Publishers, 1st edition, 2004.

[18] Symbian Ltd. Symbian OS - the mobile operating system. Se `http://www.symbian.com/`.

[19] Gianluigi Me. Exploiting buffer overflows over Bluetooth: the Bluepass tool. In *Proceedings of the Second IFIP International Conference on Wireless and Optical Communications Networks*, pages 66–70, March 2005. WOCN 2005.

[20] Sun Microsystems. Java native interface: Programmer's guide and specification. Se `http://java.sun.com/docs/books/jni/`.

[21] Sun Microsystems. Java technology. Se `http://java.sun.com`.

[22] Brent A. Miller and Chatschik Bisdikian. *Bluetooth Revealed*. Prentice-Hall, 1st edition, 2001.

[23] OpenOBEX. start [openobex]. Se `http://www.openobex.org`.

[24] Bruce Potter. Bluetooth "vulnerabilities". *Network Security*, 2:4–5, February 2004.

[25] Bruce Potter. Warchalking and Bluejacking - Myth or reality. *Network Security*, 1:4–5, January 2004.

[26] Bluetooth Special Interest Group (SIG). Bluetooth.com — Bluetooth SIG Adopter Members. Se `http://www.bluetooth.com/Bluetooth/SIG/Directory/Adopters/`.

[27] Bluetooth Special Interest Group (SIG). Bluetooth.com — the Official Bluetooth Wireless Info Site. Se `http://www.bluetooth.com/`.

[28] Bluetooth Special Interest Group (SIG). The Official Bluetooth Membership Site. Se `https://www.bluetooth.org/`.

[29] SourceForge. Sourceforge.net: Axis Openbt Stack. Se `http://sourceforge.net/projects/openbt/`.

[30] Frank Stajano and Ross Anderson. The Resurrecting Duckling: Security Issues in Ad-hoc Wireless Networks. In *Security Protocols, 7th International Workshop Proceedings*, pages 172–194. Springer-Verlag, 1999.

[31] Symantec. Symantec Corp. Se `http://www.symantec.org`.

[32] Webopedia. What is snarf? - a Word Definition From the Webopedia Computer Dictionary. Se `http://www.webopedia.com/TERM/S/snarf.html`.

[33] Wikipedia. Quality of service - Wikipedia, the free encyclopedia. Se `http://en.wikipedia.org/wiki/Quality_of_service`.

[34] Palo Wireless. Bluetooth Resource Center. Se `http://www.palowireless.com/bluetooth/`.

[35] Ford-Long Wong and Frank Stajano. Location Privacy in Bluetooth. In *Proceedings of Security and Privacy in Ad-hoc sensor Networks: Second European Workshop (ESAS 2005)*, volume 3813, pages 178–188. Springer-Verlag, 13–14 Jul 2005.

Bilaga A

Utbyggnad av Bluetooth-sniffern

Det bör vara tämligen enkelt att bygga ut Bluetooth-sniffern för en utomstående person. Vi ska nu beskriva ett exempel på hur en användare utökar vårt bibliotek med funktionen "void fooBar() ... " och kopplar den till det grafiska gränssnittet.

- Tillägg till det grafiska gränssnittet: deklarera "public native void fooBar();" i klassen BTSniffer.java. Därefter bör användaren göra ett anrop från det grafiska gränssnittet till btSniffer.fooBar().

- Skapande av header-fil: användaren kompilerar BTSniffer.java med sin Java-kompilator, därefter måste användaren utföra kommandot "javah -jni BTSniffer". Filen BTSniffer.h kommer då att skapas.

- Implementation av funktionen i bibliotekt: användaren måste först titta hur Java anropar funktionen genom att öppna filen, denne kommer då att bland annat se

```
<BTSniffer.h>
...
JNIEXPORT void JNICALL Java_BTSniffer_fooBar (JNIEnv *, jobject);
...
```

Därefter måste då användaren implementera funktionen i filen libsniffer.c. Om användaren vill följa vår syntax placerar denne funktionen längst ned i filen och låter JNI-anropen vara rena skal som i sin tur anropar “void fooBar()”. Då skulle det exempelvis se ut enligt följande:

```
<libsniffer.c>
#include "../Java/BTSniffer.h>
...
...
JNIEXPORT void JNICALL Java_BTSniffer_fooBar (JNIEnv *, jobject) {
fooBar();
}
```

Kompilering av biblioteket skiljer sig mellan olika C-kompilator och utelämnas därför.