VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ANALÝZA ZABEZPEČENÍ BEZDRÁTOVÝCH SÍTÍ S VYUŽITÍM MOBILNÍHO ZAŘÍZENÍ

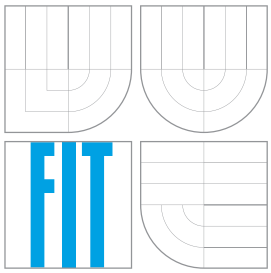
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ LÍŠKA

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ANALÝZA ZABEZPEČENÍ BEZDRÁTOVÝCH SÍTÍ S VYUŽITÍM MOBILNÍHO ZAŘÍZENÍ

ANALYSIS OF SECURITY FOR WIRELESS NETWORKS USING MOBILE DEVICE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — TOMÁŠ LÍŠKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. LUKÁŠ ARON
SUPERVISOR

BRNO 2014

Abstrakt

Tato bakalářská práce se zabývá analýzou bezpečnosti bezdrátových sítí pomocí mobilního zařízení, u kterého je aplikace tvořena pro operační systém Android. Aplikace spouští promiskuitní mód síťové karty a sleduje dostupné sítě v dosahu. Síť je následně možné podrobit bezpečnostní analýze. Práce seznamuje čtenáře s nutnou teorií bezpečnosti bezdrátových sítí, návrhem a implementací aplikace. Práce taktéž popisuje problémy spojené s touto analýzou a shrnuje dosažené výsledky.

Abstract

This bachelor's thesis deals with analysis of security for wireless networks using mobile device, where application is created for Android operating system. Application launchs promiscuous mode for wireless card and scans wireless networks in range. There is a possibility to analyze security of these networks. This work describes wireless networks security theory, proposal and implementation. This work also describes problems with analysis and evaluates the final results.

Klíčová slova

Android, Java, bezpečnost bezdrátových sítí, WEP, WPA, WPA2

Keywords

Android, Java, security of wireless networks, WEP, WPA, WPA2

Citace

Tomáš Líška: Analýza zabezpečení bezdrátových sítí
s využitím mobilního zařízení, bakalářská práce, Brno, FIT VUT v Brně, 2014

Analýza zabezpečení bezdrátových sítí s využitím mobilního zařízení

Prohlášení

Prohlašuji, že jsem tuhle bakalářskou práci vypracoval samostatně pod vedením pana Ing.Lukáš Aron

.....
Tomáš Líška
20. května 2014

Poděkování

Chtěl bych poděkovat svému vedoucímu Ing. Lukášovi Aronovi za vedení práce, odbornou pomoc, konzultace, ochotu a mnoho cenných rad při psaní téhle práce. Také bych chtěl poděkovat své přítelkyni za pevné nervy a probděné noci.

Obsah

Kapitola 1

Úvod

V dobe rozmachu bezdrôtových sietí je stále podstatnejšia a žiadanejšia otázka zabezpečenia týchto sietí. Táto otázka neplatí už iba pre firemný sektor, ale čoraz viac je podstatná aj v domácich či verejných bezdrôtových sieťach. Práca sa zaoberá analýzou bezpečnosti bezdrôtových sietí, kde je analýza realizovaná pomocou mobilnej aplikácie na platforme Android. Výpočtový výkon mobilných zariadení stále stúpa a tak veci, ktoré boli kedysi možné iba na počítačoch sa dnes dostávajú aj do našich vreciek. To okrem výhod prináša aj nevýhody a to prevažne zvýšené riziko nedostatočného zabezpečenia sietí, ku ktorým sa pomocou mobilného zariadenia pripájame a s tým spojené možné riziko odcudzenia a následného zneužitia dôležitých, či už firemných alebo osobných údajov. Spojením mobility a výkonu mobilných zariadení vzniká aplikácia, pomocou ktorej je možné analyzovať riziká bezdrôtových sietí v podstate kedykoľvek a kdekoľvek.

V kapitole 2 rozoberám jednotlivé typy zabezpečenia, ich princípy, spôsoby komunikácie a v neposlednom rade ich nedostatky a s tým spojené možné útoky. Podrobnejšie sa venujem hlavne typu WEP, ktorý je pomerne ľahko prelomiteľný a existuje viacero spôsobov ako to dosiahnuť. Ďalej sa v kapitole 3 venujem platforme Android, ktorá je čoraz populárnejšia a na ktorú je cielená aj táto aplikácia. Postupne predstavím vývojové prostredie, v ktorom bude aplikácia vyvíjaná, možnosti, ktoré toto prostredie ponúka a v krátkosti aj alternatívne programovacie jazyky a vývojové prostredia. Kapitola 4 zhodnocuje existujúce riešenia a predstavuje návrh vlastnej aplikácie. V tejto časti práce budú ukázané jednotlivé obdobia vývoja a zmeny aké sa počas vývoja uskutočňovali. V nasledujúcej časti bakalárskej práce (kapitola 5) bude už popísaná konkrétna implementácia aplikácie. Spomína sa tu aké princípy, knižnice a ďalšie potrebné komponenty boli použité v mojej aplikácii. Ďalej popisujem zariadenia, na ktorých bola aplikácia testovaná a stručnú štatistiku testov a ich úspešnosti. V poslednej kapitole nazývanej záver 6 je popis dosiahnutých výsledkov, zhodnotenie práce a možnosti či návrhy zmien v budúcnosti.

Kapitola 2

Bezpečnosť bezdrôtových sietí

Pre zabezpečenie bezdrôtových sietí existuje viacero možností. Medzi najjednoduchšie, ale aj najmenej bezpečné patria napríklad filtrovanie MAC adries alebo skrývanie SSID. Tieto spôsoby rozhodne nemôžeme považovať za bezpečné, aj keď v niektorých prípadoch to stačí na to, aby to útočník vzdal a našiel si jednoduchšiu obeť. Medzi ďalšie spôsoby patria protokoly WEP a WPA, ktoré sú síce bezpečnejšie ako predtým spomínané možnosti, no nie neprelomitelné. Protokol WEP dnes už v podstate nie je problém prelomiť a skúsený útočník by nemal mať väčší problém ani s protokolom WPA. Zatiaľ najbezpečnejší spôsob je zabezpečenie pomocou protokolu WPA2. V tejto kapitole budú postupne popísané všetky základné vyššie spomínané zabezpečenia, ich výhody a nevýhody a taktiež možné útoky.

2.1 Service Set IDentifier

SSID (Service Set IDentifier) patrí medzi základné mechanizmy zabezpečenia v protokole 802.11. Jedná sa o názov siete. Jeho dĺžka je 0 až 32 oktetov. Implicitné nastavenie je také, že prístupový bod vysiela svoje SSID (typicky každých niekoľko ms) v správe *beacon* a tým oznamuje okoliu, že je prítomný. Množstvo prístupových bodov umožňuje ukrytie SSID. Toto spôsobí, že v správach *beacon* sa na mieste SSID nachádza prázdny reťazec. Bez poznania tohto identifikátoru nie je možné sa na sieť asociovať. SSID sa vysiela v otvorenej forme v rôznych rámcoch ako napríklad *Probe Request* alebo *Probe Response*, ale aj *Association Request* a *Reassociation Request*. Niektoré z týchto správ sa vyskytujú menej často, ale napr. pri prechádzaní stanice z jednej WLAN do druhej (napr. kvôli slabému signálu) vysiela stanica *Probe Request*, na ktorú dostane odpoveď vo forme *Probe Response* od všetkých prístupových bodov v dosahu a táto odpoveď povinne obsahuje SSID. Prístupový bod môže byť nakonfigurovaný aj tak, aby *beacon* rámce s SSID nevysielal a tým schovať sieť pred bežnými užívateľmi, ale pred útočníkom to nestačí. Ten môže jednej z aktívnych staníc poslať jednoduchú požiadavku na odpojenie a to následne spôsobí, že stanica sa musí znova pripojiť pomocou správ *Probe* a *Associate* a tak vie útočník zistiť SSID prístupového bodu. Útoky teda môžeme rozdeliť na dva typy a to aktívne a pasívne. Pri pasívnom útoku útočník čaká, monitoruje sieť a snaží sa zachytiť niektorý z rámcov, v ktorom je SSID priamo viditeľné. Naopak počas aktívneho útoku útočník využíva skutočnosť, že management rámce nie sú nijakým spôsobom zabezpečené a tak poslaním falošného disasociačného alebo deautentifikačného rámca donútime stanicu sa opäť asociovať, čím nám prezradí svoje SSID.

Okrem SSID sa používa aj ESSID (Extended Service Set IDentifier), ktorý slúži ako

jedna zo základných techník pre riadenie prístupu do WLAN. ESSID je naprogramovaná hodnota do prístupového bodu pre identifikáciu siete, v ktorej sa prístupový bod nachádza. Keďže sa ESSID nevysiela, pripojenie je povolené iba autorizovaným staniciam. Sieť, ktorá používa ESSID sa označuje ako uzavretá.

Niektoré prístupové body vyššej triedy umožňujú podporu viacerých SSID. To umožňuje vytvárať tzv. bezpečnostné profily. Napríklad verejné SSID bude používať iné bezpečnostné pravidlá ako privátne SSID. Ďalšia výhoda je, že nie je treba inštalovať duplicitné zariadenia ako sa to dialo v minulosti. Informácie pochádzajú z [9] a [11].

2.2 Filtrovanie Media Access Control adries

MAC (Media Access Control) adresa je výrobcom stanovená adresa pre každú sieťovú kartu. Každá MAC adresa je jedinečná a skladá sa z 12 hexadecimálnych čísiel. Prvá polovica obsahuje ID číslo výrobcu karty a druhá polovica reprezentuje sériové číslo priradené od výrobcu. Aj keď je každá MAC adresa jedinečná, množstvo z bezdrôtových kariet umožňuje softvérovú zmenu tejto adresy, čo spôsobuje určité problémy. Pre zistenie povolenej MAC adresy existujú dva prístupy. Prvý využíva skutočnosť, že zdrojová aj cieľová adresa sa posielajú nešifrovane (aj pri použití WEP) a tak môže útočník jednoducho odpočúvať hodnoty povolených MAC adries a potom zmeniť tú svoju na hodnotu takú, aby bola platná. Ešte elegantnejšie, aj keď v niektorých prípadoch zdĺhavé riešenie, je počkať si, kým sa používateľ odpojí zo siete sám. Ak sa karta tvári ako karta s povolenou MAC adresou, prístupový bod bude presvedčený, že sa jedná o legitímnu prevádzku. Okrem rizika falšovania MAC adries sa vo väčších sieťach stáva administrácia zoznamu povolených adries veľmi zložitá a neudržateľná. Z vyššie spomínaných dôvodov preto nie je veľmi vhodné používať filtrovanie MAC adries ako jediný bezpečnostný prvok aj keď opäť platí, že v niektorých prípadoch, kedy sa nejedná o veľké siete, to útočníka odradí a nájde si ľahší cieľ. Po zistení platných MAC adries nie je pre útočníka problém svoju MAC adresu zmeniť a tak sa dostať do zabezpečenej siete, v ktorej môže následne odpočúvať komunikáciu ďalších členov. Čerpané z [9].

2.3 Wired Equivavelnt Privacy Protokol

WEP (Wired Equivavelnt Privacy) bol vyvíjaný s úmyslom poskytnúť zabezpečenie pre bezdrôtové siete na úrovni bezpečnosti drôtových sietí, ale očakávania neboli naplnené. Mal slúžiť k autentifikácii a k šifrovaniu prenášaných dát. Jedným z najväčších problémov protokolu WEP v štandarde 802.11 je to, že štandard nerieši správu kľúčov. Tajné heslo, ktoré je zdielané, musí byť distribuované všetkým užívateľom, protokol však nedefinuje akým spôsobom to realizovať. Ďalším problémom je zmena kľúča. Kľúč je treba zmeniť napr. vždy ak spoločnosť opustí nejaký zamestnanec alebo pri strate notebooku či mobilného zariadenia. V tejto kapitole budú popísané jednotlivé časti WEP protokolu ako autentifikácia, šifrovanie alebo integrita dát. Vysvetlením fungovania protokolu zistíme taktiež slabiny a následné možné útoky na daný protokol.

Autentifikácia

Autentifikácia prebieha otvorene (open system) alebo na základe zdieľaného kľúča (shared key). Za implicitné nastavenie sa považuje otvorená autentifikácia. V tomto prípade sa môže akýkoľvek klient pridružiť k prístupovému bodu. Otvorená autentifikácia prebieha

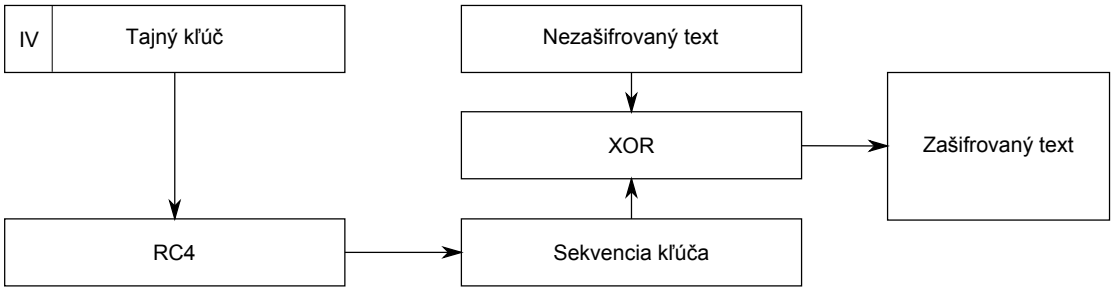
v dvoch krokoch. V prvom kroku klient vysiela autentifikačný rámec, ktorý obsahuje jeho identifikačné údaje. V druhom kroku prístupový bod alebo iný klient skontroluje identitu a vyšle späť rámec *authentication verification*. V druhom prípade, pri autentifikácii zdielaným kľúčom sa používa 40bitový statický používateľský kľúč, ktorý je rovnaký pre všetkých užívateľov danej siete. Jednou z hlavných slabín autentifikácie je skutočnosť, že v rámci WEP sa overuje totožnosť sieťovej karty a nie samotnej osoby používateľa. Autentifikácia prebieha prostredníctvom komunikácie výzva-odpoveď, kedy na základe požiadavku vyslaného klientom mu prístupový bod pošle text, ktorý klient musí zašifrovať svojim kľúčom a odoslať späť prístupovému bodu. Ten prijatú hodnotu porovná s výsledkom svojho výpočtu a pokiaľ sú obe hodnoty rovnaké, všetko prebehlo v poriadku a prístupový bod pridruží klienta do WLAN. Keďže sa prenáša najprv otvorený text, ktorý sa vzápätí prenáša aj v šifrovanej forme, hrozí riziko odhalenia kľúča útočníkom. Ďalším problémom je, že autentifikácia prebieha iba jednostranne a nie vzájomne. Klient nemá možnosť žiadať prístupový bod o to, aby sa autentifikoval.

Šifrovanie

WEP používa symetrický postup šifrovania, kedy sa pre šifrovanie a dešifrovanie používa rovnaký algoritmus a rovnaký kľúč. Šifrovanie prenášaných dát medzi klientom a prístupovým bodom sa vykonáva 64bitovým alebo 128bitovým kľúčom, ktorý je zložený z užívateľského tajného kľúča v dĺžke 40 resp. 104 bitov a dynamicky sa meniaceho inicializačného vektoru IV (Initialization Vector), vždy s dĺžkou 24 bitov. Inicializačný vektor generuje vysielacia strana, ktorá ho používa pre vytvorenie šifry a súčasne ho posiela v otvorenej forme ako súčasť záhlavia každého paketu. Príjemca použije inicializačný vektor IV pre spojenie so zdielaným WEP kľúčom a prevedie dešifrovanie prijatých dát. Inicializačný vektor sa používa ako zmiernenie statickosti WEP kľúča, kedy by rovnaká správa viedla k rovnako zašifrovanému textu a tým by mal útočník všetky podklady pre jednoduché zistenie kľúča. Množina možných identifikačných čísiel je 2^{24}, čo znamená približne 17 miliónov. Problém je, že norma nešpecifikuje ako sa má inicializačný vektor (IV) meniť a ani to, či sa má meniť s každým paketom. Väčšina výrobcov mení inicializačný vektor s každým paketom, a veľakrát je dokonca verejne známy vzorec nastavenia napr. pri inicializácii karty vždy začať na 0 a s každým paketom zvyšovať o 1. Tento spôsob bohužiaľ veľmi uľahčuje prácu útočníkovi. WEP ďalej používa symetrickú prúdovú šifru RC4, ktorá je taktiež využívaná napr. v SSL. Pre zabezpečenie WEP bola zvolená z dôvodu jednoduchosti implementácie. Táto prúdová šifra RC4 umožňuje vytvoriť z kľúča šifrovací prúd (cipher stream) tak, aby bolo možné šifrovanie textu s ľubovoľnou dĺžkou. Pre WEP bola zvolená dĺžka kľúča 40 bitov aj keď v skutočnosti RC4 povoľuje kľúč s dĺžkou až 256 bitov. RC4 pracuje ako generátor pseudonáhodných čísiel, kde základ je kombinácia tajného kľúča a inicializačného vektoru IV. Výsledná postupnosť sa pre zašifrovanie spojí s dátami (textom) pomocou logickej funkcie XOR. Obrázok 2.1, prevzatý z [9], zobrazuje ukážku šifrovania protokolom WEP, kde inicializačný vektor IV spolu s tajným kľúčom tvoria RC4 kľúč. Tento kľúč je pomocou logickej funkcie XOR spojený s výslednými dátami (nezašifrovaný text) a vzniká nám tak text zašifrovaný.

Zabezpečenie integrity dát

Pre zaistenie integrity dát sa vykonáva kontrolný súčet dátovej časti rámca, ktorého výsledkom je ICV (Integrity Chceck Value), ktorý sa pripája na koniec rámca. Hodnota ICV je zašifrovaná spolu s dátami. Ak u príjemcu po dešifrovaní nesúhlasí ICV s uvedenou



Obrázek 2.1: Šifrovanie vo WEP protokole

hodnotou v rámci, rámec sa zahodí. Kontrolný súčet funguje pomocou funkcie cyklického kontrolného súčtu (CRC-32). Ide o lineárny blokový kód, ktorý ľahko podlieha útokom, kde sa rôznymi spôsobmi zamieňajú bity v rámci, bez toho aby sa zmenil kontrolný súčet. Zabezpečenie integrity v protokole WEP nie je veľmi bezpečné. Informácie boli nájdené v [11].

Slabiny a útoky

Medzi hlavné slabiny WEP patrí, že protokol nešpecifikuje, ako má byť implementované generovanie inicializačného vektoru IV. Ako už bolo spomenuté, pre inicializáciu šifry RC4 sa používa kombinácia inicializačného vektoru IV a tajného kľúča. Dôvodom prečo sa inicializačný vektor používa je, že potrebujeme zaistiť, aby bola inicializačná hodnota generátoru stále iná. Základný požiadavok šifry RC4 je, aby sa za žiadnych okolností znovu nepoužila rovnaká inicializačná hodnota. S každým odoslaným paketom potrebujeme generátor inicializovať na inú hodnotu, to znamená, že pri vyšších prenosových rýchlostiach vyčerpáme celý 24 bitový priestor pomerne rýchlo a tým porušujeme najhlavnejšie pravidlo RC4, ktoré zakazuje použiť kľúč opakovane. Akonáhle nastane situácia, že sa inicializačný vektor zopakuje hovoríme, že prišlo ku kolízii a útočník môže z nazbieraných informácií dešifrovať dáta. Ďalej vie útočník využiť vlastnosti RC4 k tomu, aby zaslal klientovi nejaký text a odpočúvať ako bude tento text zašifrovaný prístupovým bodom. Keďže v tomto momente útočník pozná zašifrovaný aj nezašifrovaný text, môže odvodiť šifrovací kľúč. Tento útok je však treba zopakovať, nakoľko šifrovací kľúč je platný iba pre dané inicializačné vektory. Monitorovaním siete si útočník vytvorí databázu inicializačných vektorov a rámcov, na základe ktorých môže uhádnuť používaný šifrovací kľúč. Ďalšie slabé miesto je integrita dát. ICV, ktoré zaisťuje integritu dát vo WEP nie je dostatočne spoľahlivý mechanizmus, nakoľko zámena bitov nezmení hodnotu ICV. Toto umožňuje útočníkovi vytvoriť vymyslenú správu, ktorú zašle príjemcovi a keďže táto správa nedáva žiadny zmysel, príjemca vygeneruje chybovú správu a odošle ju späť odosielateľovi. Útočník tak dokáže odhadnúť obsah správy a zo zašifrovanej podoby je schopný odvodiť šifrovací kľúč. Vďaka týmto slabinám je možné uskutočniť niekoľko typov útokov ako napríklad útoky hrubou silou - proti 128bitovému útoku sa používa slovníkový útok a proti 64bitovému zasa distribuovaný útok, ďalej sú to útoky typu FMS - FMS podľa autorov Fluhrer,Mantin a Shamir, ktorí ho popísali v roku 2001, alebo v neposlednej rade vylepšené útoky FMS - podľa návrhu H1kari a Dasb0den Labs. Pri útoku typu FMS je podľa veľkosti kľúča treba približne 6 až 8 miliónov paketov na to, aby sme boli schopný zistiť správny WEP kľúč. Pri sieťach Wi-Fi, kde je priepustnosť 812 rámcov/sekunda je možné zlomiť WEP kľúč za približne dve hodiny. Problémom útoku FMS je potreba získania veľkého objemu dát, čo znamená, že v niektorých prípadoch by to

trvalo niekoľko dní, dokonca týždňov. Tento problém čiastočne rieši vylepšený útok FMS, ktorý skracuje dobu prelomenia kľúča približne o 1/20 a stačí mu iba približne 500 tisíc paketov. Treba mať ale na pamäti, že útoky typu FMS závisia na schopnosti zachytávať tzv. slabé kľúče. Mnoho hardvérových výrobcov postupne implementovalo firmware aktualizáciu, ktorá jednoducho preskakuje špecifické inicializačné vektory, ktoré spôsobujú tieto slabé kľúče. Toto spôsobí, že útoky typu FMS sú neefektívne a neúčinné. Toto je taktiež jeden z dôvodov, prečo je aktualizácia firmware tak dôležitá. Informácie pochádzajú zo [7], [5], [3], [2].

2.4 Wi-Fi Protected Access Protokol

WPA (Wi-Fi Protected Access) vznikol ako dočasné riešenie, kým bude schválený bezpečnostný doplnok normy IEEE 802.11i. Hlavným komponentom protokolu 802.11i, ktorý v tej dobe ešte nebol dokončený, bola šifra AES. WPA je akýsi medzičlánok zabezpečenia WLAN: je spätne kompatibilný s WEP a dopredu zlúčiteľný s 802.11i/WPA2, ktorý si rozoberieme neskôr. Z tohto tiež vyplýva, že ak sa v sieti nachádzajú produkty s WEP aj s WPA, použije sa slabšie WEP. Protokol WPA z dôvodu spätnej kompatibility používa rovnaký šifrovací mechanizmus RC4 ako WEP. Avšak protokol použitý vo WPA má vyššiu zložitosť a to má vplyv na výkonnosť : v porovnaní s WEP sa znižuje výkonnosť o 5–15%. V tejto kapitole si postupne popíšeme, podobne ako v prípade WEP, jednotlivé časti protokolu WPA ako autentifikácia, šifrovanie alebo integritu dát. Aj keď je protokol WPA bezpečnejší ako jeho predchodca WEP, nevyhol sa ani tento protokol rôznym chybám a slabinám. Niektoré z týchto slabín sa dajú využiť na útoky, ktoré tu spomeniem.

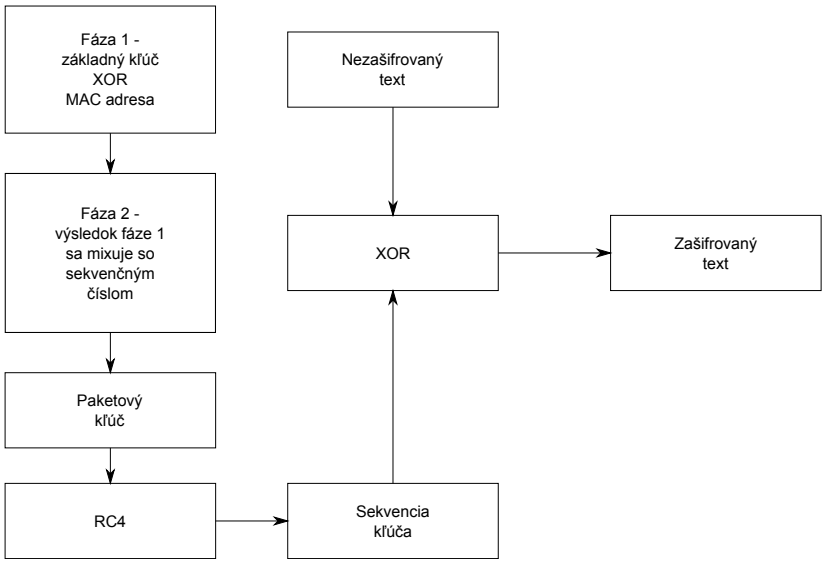
Autentifikácia

Oproti autentifikácii v protokole WEP došlo v prípade WPA ku zlepšeniu. WPA ponúka niekoľko režimov autentifikácie. Vo firemnom prostredí sa predpokladá využitie centrálneho autentifikačného serveru, ktorý je zodpovedný za distribúciu kľúčov, typicky je týmto serverom RADIUS. Naopak v domácom prostredí sa častejšie používa jednoduchší režim prednastaveného kľúča PSK (Pre-Shared Key). Wi-Fi Alliance doporučuje použitie EAP-TLS (Extensible Authentication Protocol - Transport Level Security) pre všetky produkty s podporou WPA. EAP-TLS nepodporuje iba autentifikáciu, ale aj odvodzovanie kľúčov. Autentifikácia klienta a serveru prebieha vzájomne pomocou digitálnych certifikátov. EAP-TLS je zložitá metóda na implementáciu, ale zatiaľ nemá žiadnu známu bezpečnostnú slabinu. Úspešnou autentifikáciou získame hlavný kľúč PMK (Pairwise Master Key), z ktorého sa ďalej odvodí ďalších päť kľúčov potrebných pre ďalšie použitie. V prípade nepoužitia 802.1x, dynamické kľúče sa nahradia zdielaným kľúčom PSK. Možnosť offline útokov vzniká pri sietiach, ktoré nepoužívajú centralizovaný autentifikačný server ako napríklad RADIUS. Ďalším problémom je, že sa v týchto sieťach často volia veľmi jednoduché heslá, ktoré je možné zistiť slovníkovým útokom. Informácie môžme nájsť v [5] alebo [12].

Šifrovanie

Z dôvodu zlepšenia šifrovania sa v protokole WPA zavádza mechanizmus TKIP (Temporal Key Integrity Protocol), ktorý používa tri hlavné prvky a to lepšiu kontrolu integrity dát nazvanú Michael (MIC - Message Integrity Code), ďalej funkciu, ktorá mixuje kľúč pre každý paket a taktiež vylepšené pravidlá generovania inicializačného vektoru. Mechanizmus

TKIP predlžuje dĺžku správy o 12 bajtov, kde 8 bajtov je pre kód integrity správy (MIC) a 4 bajty pre rozšírenú informáciu inicializačného vektoru. WPA tiež rozširuje dĺžku inicializačného vektoru na 48 bitov oproti 24 bitovému IV vo WEP šifrovaní. Toto zamedzuje častému opakovaniu kľúčov, ako to bolo v prípade WEP. V skutočnosti sa hodnota inicializačného vektoru delí na dve časti. Prvá časť má 16 bitov a doplní sa do 24 bitov pre tradičný IV a druhá 32 bitová časť poslúži ako poradové číslo paketu, ktoré je následne použité pri mixovaní kľúčov. TKIP tiež ponúka tzv. sekvenčné počítadlo. Toto počítadlo umožňuje, že inicializačný vektor sa vďaka nemu zvyšuje postupne a tým sa všetky pakety mimo postupnosť zničia, čím sa zabráni útokom typu *replay*. TKIP si pamätá posledných 16 hodnôt inicializačného vektoru IV a kontroluje, či do nich vyslaný rámec pasuje. Ak rámec nebol doteraz prijatý a pasuje do postupnosti, tak je prijatý. Na obrázku 2.2 je názorne zobrazené šifrovanie protokolom TKIP vo WPA. Zašifrovaný text vzniká po niekoľkých fázach. Vo fáze jedna sa pomocou logickej funkcie XOR spojí MAC adresa spolu so základným kľúčom. Výsledok fáze jedna sa v ďalšej fáze mixuje so sekvenčným číslom, ktorého výsledok je paketový kľúč. Ten následne vstupuje do algoritmu RC4, z ktorého vyjde sekvencia kľúča, ktorá sa spojí pomocou logickej funkcie XOR spolu s nezašifrovaným textom a tým dostávame zašifrovaný text. Obrázok pochádza z literatúry [9].



Obrázek 2.2: Šifrovanie protokolom TKIP vo WPA, prevzaté z [9]

Integrita dát

Ako bolo spomenuté vyššie, pre zaistenie integrity dát vo WPA je použitý mechanizmus MIC alias Michael (Message Integrity Code). Tento mechanizmus pridáva digitálny podpis ku každému rámcu, čím zamedzuje útoky typu *man-in-the-middle*. Tento digitálny podpis je automaticky vypočítaný a zabudovaný do dátovej časti rámca. Výpočet digitálneho podpisu prebieha na základe dátovej časti rámca, zdrojovej a cieľovej MAC adresy, poradového čísla paketu a náhodnej hodnoty. Kontrola samotného MIC na strane príjemcu prebieha až po kontrole hodnôt IV, CRC a ICV. Ak je teda MIC v neporiadku, tak je takmer isté, že sa jedná o aktívny útok. MIC bol navrhnutý tak, aby aj adaptéry s menším procesorovým výkonom nemali problém s aktualizáciou a následnou prevádzkou. Toto spôsobuje, že aj

keď je MIC lepší ako pôvodne používaný CRC, tak nedosahuje také zabezpečenie, ako by sa v tejto kategórii očakávalo. Tento mechanizmus je odolný voči útokom, ktoré boli úspešné u ICV, ako napríklad falšovanie záhlavia alebo zámena bitov. Avšak neodolá niektorým útokom typu DoS.

Slabiny a útoky

aj napriek tomu, že oproti WEP ponúka WPA zlepšenie takmer v každej oblasti, bolo len otázkou času kedy sa nájde zraniteľné miesto. Jedno z nich je TKIP, ktoré nie je odolné voči útokom typu vstrekovania paketov. Ďalším možným útokom je slovníkový útok. Tento slovníkový útok je možný vďaka slabým, krátkym, zle vybraným heslám vytvoreným užívateľom. Našťastie je jednoduché aj riešenie. Podľa Roberta Moskowitza je treba zvoliť kľúč s dĺžkou aspoň 96 bitov alebo prístupové heslo, ktoré obsahuje skomoleninu s dĺžkou aspoň 20 bitov. Ďalším možným útokom na TKIP je keystream recovery attack. Tento útok je rozšírenie chop-chop útoku praktizovaného vo WEP protokole. WEP používal kryptograficky nie bezpečný mechanizmus CRC-32, kde útočník mohol hádať jednotlivé bajty a prístupový bod potvrdil alebo zamietol správnosť tohto pokusu. Ak bol typ správny, útočník to vedel zistiť a mohol pokračovať v hádaní ďalej. V prípade TKIP však musí útočník čakať aspoň 60 sekúnd po neúspešnom pokuse, kým môže v útoku pokračovať. Tento útok sa tiež nazýva Beck-Tews útok.

2.5 Wi-Fi Protected Access II Protokol

WPA II (Wi-Fi Protected Access II) alebo tiež 802.11i je dodatok k IEEE 802.11, ktorý bol schválený v roku 2004 a vylepšuje autentifikačný a šifrovací algoritmus pre bezdrôtové siete. Hlavný komponent protokolu WPA II je šifra AES (Advanced Encryption Standard), ktorá ponúka rôzne režimy a v prípade 802.11i používa čítačový režim s protokolom CBC-MAC (CCM) označovaný aj ako AES-CCMP. AES-CCMP nahradzuje prúdovú šifru RC4 použitú v protokoloch WEP a WPA. Táto šifra je natoľko bezpečná, že odpovedá americkému federálnemu štandardu FIPS (Federal Information Processing Standarts). Rovnako ako RC4 aj AES je šifra so symetrickým kľúčom, takže šifrovanie aj dešifrovanie textu prebieha rovnakým zdielaným tajným kľúčom. Avšak narozdiel od RC4, AES pracuje s blokmi, ktoré majú veľkosť 128 bitov a preto ju značíme ako blokovú. WPA II sa zameriava hlavne na utajenie dátových rámcov či autentifikáciu a naopak nerieši ochranu management rámcov, ochranu proti DoS útokom, alebo ochranu proti útokom na vyšších vrstvách ako napríklad ARP spoofing.

Autentifikácia a šifrovanie

Autentifikácia WPA II podobne ako vo WPA ponúka dva režimy, pre podniky a pre osobné využitie. V prípade podnikového režimu sa jedná o plnú podporu WPA II vrátane PSK (Pre-Shared Key) a 802.1x. V prípade nastavenia pre osobné použitie sú požiadavky na zabezpečenie menšie, takže nie je nutné zavádzať 802.1x a zostáva iba PSK. Pre silnejšie šifrovanie sa vo WPA II používa CCMP (Counter Cipher Mode with Block Chaining Message Authentication Code Protocol) a protokol TKIP používaný vo WPA je už iba voliteľný. CCMP narozdiel od WEP používa dynamické regenerovanie kľúčov a veľkosť kľúča je 128 bitov. Zaisťuje naraz autenticitu, utajenie, kontrolu integrity dát (MIC s dĺžkou 64 bitov) a číslovanie paketov na ochranu proti replay útokom. CCMP však predlžuje dátový rámec

o 16 bytov. V CCMP sa v režime CCM používa AES (Advanced Encryption Standard) pre šifrovanie prenášaných dát. Podobne ako TKIP, CCMP používa 48 bitový inicializačný vektor IV nazývaný aj ako PN (packet number) a variáciu MIC. Dĺžka 48 bitov by mala zaistiť vyhnutie sa opätovnému používaniu inicializačných vektorov IV. Ako už bolo spomenuté AES je dostatočný šifrovací mechanizmus aj pre vládne účely a je založený na algoritme Rijndael a môže používať kľúče s veľkosťou 128, 196 alebo 256 bitov. AES taktiež umožňuje šifrovanie a dešifrovanie robiť paralelne. Generovanie kľúčov pre každý paket nie je vďaka sile AES potrebné. Preto CCMP používa relačný kľúč pre šifrovanie dát a generovanie kontrolného súčtu, ktorý má u CCMP dĺžku 8 bitov a je teda o mnoho silnejší ako MIC u TKIP.

Slabiny a útoky

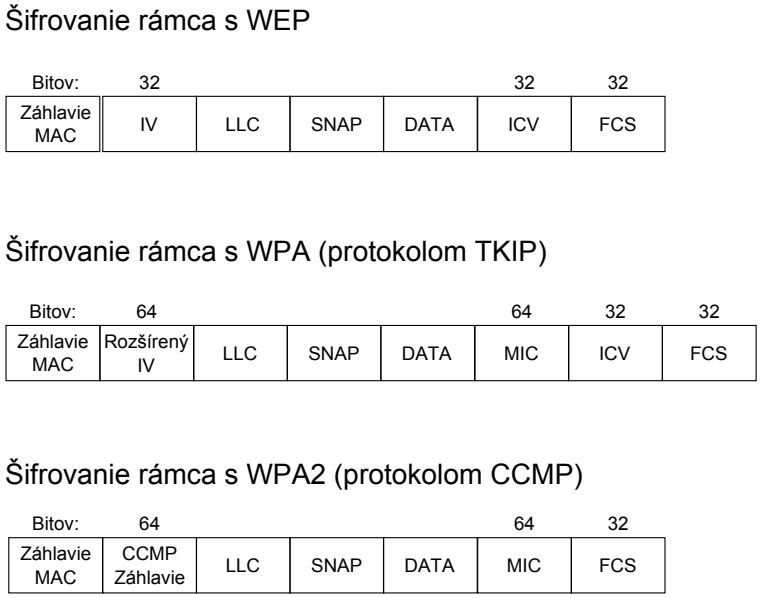
Aj keď patrí WPA II medzi momentálne najbezpečnejší protokol, vedci z AirTight našli chybu, ktorá umožňuje čiastočné zníženie bezpečnosti. Chyba bola nazvaná "Hole196". Hlavným dôvodom k tejto chybe je GTK (group temporal key), ktorý je zdielaný medzi všetkými autorizovanými klientmi. V štandardných prípadoch iba prístupový bod má možnosť vysielať skupine adresované dáta zakódované pomocou GTK a klienti sú schopní dekódovať dáta pomocou GTK. Avšak nič neobmedzuje autorizovaného klienta od možnosti vstreknutia falošných GTK paketov. Zneužitie tejto chyby povoľuje autorizovanému užívateľovi (útočníkovi) získať a dekódovať dáta od autorizovaných užívateľov, skenovať zraniteľnosť ich wi-fi zariadení, inštalovať malware a podobne. Táto chyba je však napadnuteľná iba z vnútra, keďže útočník, v tomto prípade autorizovaný klient, potrebuje poznať GTK. Útočník môže využiť Hole196 tromi spôsobmi: ARP otrávenie a man-in-the-middle attack, vstreknutie škodlivého kódu do ostatých autorizovaných wi-fi zariadení alebo spustenie denial-of-service (DoS) útokov bez použitia odpájajúcich rámcov. Ďalšia možná slabina bola publikovaná v marci roku 2014 International časopisom Journal of Information and Computer Security. Táto štúdia ukázala, že za určitých okolností použitie programu Aircrack suite spusteného cez FPGA (Field Programmable Gate Array) umožňuje uskutočniť úspešný útok hrubou silou.

2.6 Stručné porovnanie protokolov

Rozdiely vo formáte a šifrovaní správ medzi jednotlivými protokolmi môžeme vidieť na obrázku 2.3, kde je viditeľná podobnosť medzi protokolmi WEP a WPA. Tou zmenou je 64 bitová MIC časť, zabezpečujúca integritu dát, ktorá je pridaná do WPA protokolu. Táto podobnosť je zámerná, nakoľko WPA bolo predstavené ako vylepšenie WEP protokolu, aby zariadenia používajúce WEP mohli bez problémov prejsť aj na bezpečnejší protokol WPA.

Na druhej strane, tabuľka 2.1 zobrazuje porovnanie vlastností medzi protokolmi a môžme vidieť, že autentifikácia v prípade protokolov WPA a WPA2 je rovnaká, avšak v šifrovaní nájdeme rozdiely. Oproti tomu protokol WPA II môže používať rovnakú autentifikáciu ako v prípade WPA, no pre šifrovanie sa už ďalej využíva bezpečnejší protokol AES. Zatiaľ čo väčšinu zariadení s protokolom WEP je možné aktualizáciou software prerobiť aj na zariadenia podporujúce WPA, v prípade prechodu z WPA na WPA II toto možné nie je.

Informácie boli čerpané z knihy [11].



Obrázek 2.3: Rozdiely vo formáte a šifrovaní správ medzi protokolmi

Protokol	WEP	WPA	WPA2
Autentifikácia	otvorená	EAP-TLS alebo PEAP	EAP-TLS alebo PEAP
Šifrovanie	statický WEP	TKIP/CKIP	AES

Tabulka 2.1: Porovnanie vlastností WEP,WPA a WPA2

2.7 Dostupné nástroje pre analýzu

V tejto podkapitole sa budem venovať dostupným softvérovým nástrojom pomocou ktorých je možné analyzovať rôzne bezpečnostné riziká v bezdrôtových sieťach. Týchto nástrojov je pomerne veľa, takže spomeniem iba pár najpoužívanejších. Taktiež sa pozriem na dostupné hardvérové zariadenia využívajúce tieto softwarové nástroje. Okrem zariadení dostupných ako doplnok pre notebooky či iné zariadenia, pokúsim sa zamerať hlavne na zariadenia prenosné ako sú telefóny a podobne. Každé z týchto zariadení má samozrejme svoje výhody a nevýhody, ktoré by som chcel pri každom zariadení stručne zhrnúť.

Softwarové nástroje

Jedným z najpoužívanejších nástrojov je Aircrack-ng. Jedná sa o program pre analýzu bezpečnosti hlavne protokolov WEP a WPA-PSK. Aircrack využíva útoky ako FMS s rôznymi vylepšeniami ako Korek útoky alebo tiež útoky typu PTW. Vďaka tomuto patrí k jedným z najrýchlejších a najrozšírenejších nástrojov. Aircrack využíva promiskuitný mód bezdrôtovej karty a pomocou neho dokáže monitorovať sieťové pakety. Program je možné používať v operačných systémoch Linux, Windows alebo dokonca Android či iOS.

Zariadenia

V čase písania tejto bakalárskej práce (máj 2014) vydala spoločnosť Pwnie Express svoje dve najnovšie zariadenia. Zhodou okolností sú to mobilné zariadenia. V prvom prípade,

ktorý si neskôr predstavíme detailnejšie, sa jedná o mobilný telefón Nexus 5. V druhom prípade ide zase o tablet Nexus 7. Oba tieto prístroje sú vybavené veľkým množstvom open source testovacích, sieťových či administratívnych aplikácii ako napríklad aircrack-ng, MAC changer, Airodump, Kismet, Tcpdump, netcat a mnoho ďalších. Zariadenie je dodávané s externou anténou podporujúcou siete 802.11b/g/n a vstrekovanie paketov či promiskuitný mód. Ďalšou externou súčasťou je USB-Ethernet adaptér pre testovanie bezpečnosti drôtových sietí či externá bluetooth anténa s podporou vstrekovania paketov pre testovanie bezpečnosti bluetooth zariadení. Zariadenie je vybavené custom Kali Linuxom, ktorý je bezpečnejšou, vyvinutejšou verziou BackTrack Linuxu pripravenou aj pre podnikovú sféru. S hardvérových špecifikácií môžme spomenúť 2GB RAM pamäť, 4-jadrových 2,3 GHz procesor či 2300 mAh batériu. Cena tohto zariadenia je však takmer 1300 dolárov. Na obrázku 2.4 je možné vidieť produkt spoločnosti Pwnie Express, ktorá sa zaoberá testovaním bezpečnosti sietí. V tomto prípade sa jedná o najnovší model (rok 2014), ktorý vychádza zo zariadenia LG Nexus 5.

Obrázek 2.4: Pwn Phone model 2014 vychádzajúci z LG Nexus 5

Kapitola 3

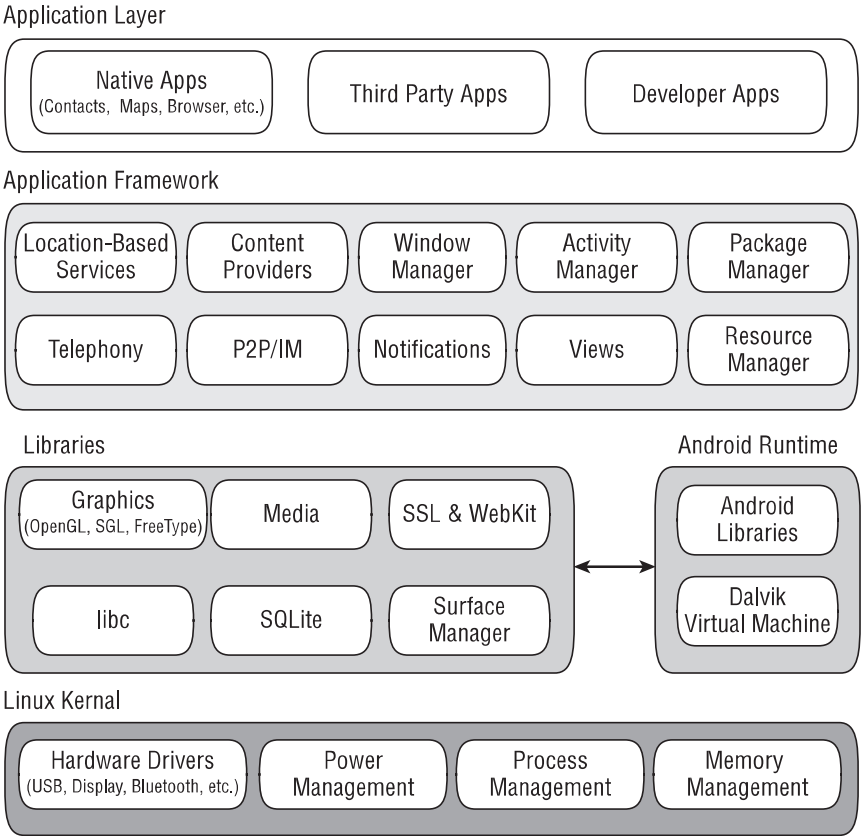
Programovanie pre platformu Android

Programovanie pre platformu Android je čoraz populárnejšie a to hlavne preto, že Android sa nachádza vo väčšine predávaných mobilných zariadení na svete. Táto rozšíriteľnosť láka okrem investorov a výrobcov samozrejme aj programátorov. Táto kapitola bude rozdelená na niekoľko častí, kde v prvej časti bude popis platformy Android, spomenuté budú aj výhody a nevýhody tohto operačného systému a programovanie aplikácií vyvíjaných pre tento systém. V ďalších častiach sa zameriam na vývojové prostredia a sadu nástrojov využívaných pri programovaní. V kapitole popíšem okrem hlavného programovacieho jazyka Java a jeho vývojové prostredie aj alternatívne vývojové prostredia a programovacie jazyky.

3.1 Android a jeho verzie

Android je open source platforma, ktorá vznikla predovšetkým pre mobilné zariadenia ako PDA, tablety, inteligentné telefóny, no čoraz častejšie sa nachádza aj v iných zariadeniach ako fotoaparáty, chladničky, automobily a ďalšie. Tento operačný systém je založený na jadre Linuxu a je vyvíjaný spoločnosťou Google a takzvanou Open Handset Alliance (OHA), ktorá združuje spoločnosti zaoberajúce sa výrobou mobilného hardwaru, softwaru a zopár telekomunikačných spoločností. Prvá beta verzia operačného systému spolu s SDK boli predstavené v roku 2007. O rok neskôr bol predstavený prvý komerčne dostupný telefón s operačným systémom Android - HTC Dream. Stručný prehľad jednotlivých verzií ako aj popis hlavných noviniek a zmien je ukázaný na obrázku A.3. Architektúra Androidu je názorne ukázaná na obrázku 3.1, ktorý bol prevzatý z literatúry [10]. Na obrázku môžme vidieť rozdelenie do štyroch vrstiev. Najnižšia vrstva (Linux kernel) je jadro operačného systému. Jadro systému Android je postavené na Linuxovom jadre. Do tejto vrstvy patria základné ovládače (napr. bluetooth,usb, wi-fi, audio a ďalšie), správa pamäti alebo správa procesov. Ďalšia vrstva (Libraries) zahŕňa knižnice, ktoré sú väčšinou napísané v C alebo C++ a sú využívané rôznymi časťami systému. Patria sem napríklad knižnice pre prácu s grafikou OpenGL, pre prácu s databázami SQLite, pre prácu s médiami a ďalšie napríklad sieťové knižnice. Na rovnakej úrovni sa tiež nachádza časť (Android Runtime) obsahujúca virtuálny stroj. V súčasnosti sa ako hlavný virtuálny stroj používa Dalvik, ktorý je špeciálne optimalizovaný pre mobilné zariadenia. V najnovšej verzii Androidu 4.4 (máj 2014) sa najnovšie nachádza aj experimentálna implementácia nového virtuálneho stroja ART, ktorý by mal v budúcnosti nahradiť Dalvik. Ďalšia vrstva (Application framework) poskytuje triedy

pre vytváranie nových aplikácií a tak je veľmi podstatná pre vývojárov. Táto vrstva tiež poskytuje určitý typ abstrakcie pre hardwardový prístup, správu užívateľského prostredia a aplikačných prostriedkov. Poslednou vrstvou je (Application Layer), ktorú tvoria aplikácie používané užívateľmi. Môže sa jednať o aplikácie tretích strán alebo už predinštalované aplikácie.



Obrázek 3.1: Architektúra Androidu

Medzi základné komponenty Android aplikácií patria:

- **Aktivity (Activity)** - odpovedá jednej obrazovke a obsahuje grafické užívateľské prostredie pre interakciu s užívateľom. Aplikácia väčšinou obsahuje viac aktivít, medzi ktorými je užívateľ schopný prepínať.
- **Služby (Service)** - neposkytuje grafické užívateľské prostredie, ale predstavuje proces bežiaci na pozadí. Väčšinou sa využíva pri dlho trvajúcich úlohách k prístupu k vzdialeným zdrojom, ako napríklad server alebo ku kontrole aktualizácií.
- **Poskytovateľ obsahu (Content Provider)** - aplikačné rozhranie pre zdieľanie dát medzi aplikáciami, ale aj pre zdieľanie dát medzi jednotlivými aktivitami. Poskytovateľ obsahu môže byť súbor, SQLite databáza, webový kanál alebo zložitejšie varianty.
- **Prijímač vysielania (Broadcast receiver)** - komponent slúžiaci k tzv. počúvaniu oznámení. Podľa určenia na ne reaguje rôznym spôsobom, ako napríklad výpisom na stavový riadok alebo spustením iného komponentu. Aplikácie môžu využívať vysielania systémové alebo vytvárať svoje vlastné.

Informácie sú dohľadateľné na [4].

Životný cyklus aplikácie a služby

Ako už bolo spomenuté vyššie, aktivita je komponent aplikácie, ktorý odpovedá jednej obrazovke s ktorou môže užívateľ komunikovať vo forme vykonávania určitých akcií, ako napríklad vytočiť číslo, spraviť fotku, poslať email alebo prezerať mapu. Každá aktivita predstavuje okno s vykresleným užívateľským prostredím. Okno typicky vypĺňa veľkosť obrazovky, ale môže byť aj menšie a plávať na vrchnej vrstve okna. Aplikácia sa väčšinou skladá z viacerých aktivít, ktoré sú medzi sebou zviazané. Typicky býva jedna z nich hlavná aktivita, ktorá je zobrazená ako úvodná obrazovka pri zapnutí aplikácie. Každá aktivita môže začať ďalšiu aktivitu a vždy keď sa tak stane, pôvodná aktivita je pozastavená a uchovaná v zásobníku. Na obrázku A.1 je popísaný životný cyklus aktivity Android aplikácie. Životný cyklus aktivity sa dá rozdeliť do troch častí. Úplný životný cyklus prebieha medzi volaním metódy **onCreate()** a volaním **onDestroy()**. Po zavolaní **onDestroy()** by mala aktivita uvoľniť všetky zdroje, ktoré používa. Ďalšou časťou je viditeľný životný cyklus. Ten prebieha medzi volaniami **onStart()** a **onStop()**. Metóda **onStop()** je volaná napríklad keď začne nová aktivita a doteraz používaná už nie je viditeľná. Metódy **onStart()** a **onStop()** môžu byť počas úplného životného cyklu volané aj viac krát. Poslednou časťou je životný cyklus na popredí. Toto sa deje medzi volaniami **onResume()** a **onPause()**. Počas tejto doby je aktivita v popredí všetkých ostatných aktivít. Aktivita môže frekventovane prechádzať medzi stavom, kedy je v popredí a stavom kedy beží na pozadí. Metóda **onPause()** sa napríklad volá vždy, keď je zariadenie uspané alebo pri objavení dialógu.

Na druhej strane služba je aplikačný komponent používajúci sa pri dlhých operáciach, ktoré prebiehajú na pozadí a nevyžadujú interakciu užívateľa. Ďalší aplikačný komponent môže spustiť službu a tá bude pokračovať na pozadí, aj keď sa používateľ prepne do inej aplikácie. Typickou službou môže byť napríklad prehrávanie hudby, vykonávanie I/O operácií či sieťová komunikácia. Obrázok A.2 zobrazuje životný cyklus Android služby. Životný cyklus služby je omnoho jednoduchší ako životný cyklus aktivity. Avšak je veľmi podstatné venovať tomu pozornosť, pretože užívateľ nemusí vedieť, že služba beží na pozadí a využíva prostriedky zariadenia zbytočne. Životný cyklus začína vytvorením a končí zničením služby a dá sa rozdeliť na dva typy : viazaná a neviazaná. Neviazaná služba je vytvorená po zavolaní metódy **startService()**. Po zavolaní služba beží po dobu neurčitú, až kým sama nezavolá **stopSelf()** alebo iný komponent zavolá metódu **stopService()**. Viazaná služba je vytvorená volaním **bindService()**. Klient komunikuje so službou pomocou rozhrania **IBinder** a môže ukončiť spojenie zavolaním **unbindService()**. Viacero klientov sa môže naviazať na rovnakú službu a táto služba bude zrušená až vtedy, keď sa odviažu všetci klienti. Informácie a obrázky boli nájdené na [1].

Výhody a nevýhody

Ako každá platforma, aj Android má svoje výhody a nevýhody. Medzi výhody môžeme rozhodne zaradiť jeho popularitu a rozšíriteľnosť vo svete. Naprogramovaná aplikácia sa tak dostane k obrovskému množstvu užívateľov a to hlavne vďaka obchodu s aplikáciami, ktorý Google nazýva Play Store. Medzi ďalšiu výhodu by som zaradil jeho otvorenosť, vďaka ktorej je teoreticky možné naprogramovať čokoľvek. Napísal som teoreticky z toho dôvodu, že k niektorým činnostiam potrebujeme, aby mal telefón prístup typu "root".

To, čo patrí medzi najväčšie výhody, patrí zároveň aj medzi jeho najväčšie nevýhody a to je jeho rozšíriteľnosť. Vďaka tomu, že je Android tak populárny vývojári sa pri programovaní aplikácie musia vysporiadať s niekoľkými aspektami. Jedným z nich je fragmentácia Androidu. Na trhu nájdeme Android od verzie 2.2 až po najnovšiu verziu 4.4. Google sa síce v poslednej aktualizácii snaží tento trend obmedziť a najnovšia verzia je optimalizovaná aj pre slabšie zariadenia, no iba čas ukáže, či sa mu to naozaj podarí. Viac o týchto zmenách bude písané v ďalšej časti tejto práce. Ďalší problém je obrovské množstvo rôznych prístrojov. To prináša starosti spojené s prispôsobením aplikácie na rôzne veľkosti a rozlíšenia displejov či rôzne procesory.

Súčasnosť

V súčasnosti je každý deň aktivovaných vyše 1,5 milióna zariadení s Androidom. Najnovšia verzia je číslo 4.4 a nesie názov Kit Kat. Kit Kat prináša niekoľko podstatných zmien, či už v dizajne alebo funkčnosti. Hlavným cieľom v najnovšej verzii bolo optimalizovanie pamäte. Android by mal odteraz bežať plynulo a bez problémov aj na zariadeniach s pamäťou RAM okolo 512 MB. Pre vývojárov sú pripravené nové nástroje pre analýzu využitia pamäte a API, na uistenie, že ich aplikácia je pamäťovo efektívna. Pridané sú aj možnosti zistenia, či aplikácia beží na zariadení s menšou pamäťou a ak áno, následná možnosť zmeniť, prípadne obmedziť niektoré funkcie. Ďalej sa pridáva podpora zobrazenia od okraja k okraju, čiže bez notifikačnej lišty alebo softvérových tlačidiel. Táto funkcia môže byť veľmi užitočná hlavne pri aplikáciach typu prehliadač pdf súborov, video prehrávač atď. Android má v najnovšej verzii zabudované aj nahrávanie obrazovky, čo umožňuje riešenie niektorých problémov formou návodov nahraných priamo telefónom. Ďalšou novinkou je tzv. Host card emulátor, ktorý umožňuje emulovať platobné karty a tým umožniť platbu pomocou mobilného zariadenia. Taktiež tu nájdeme aj podporu bezdrôtového tlačenia či nových nízko energetických senzorov. Medzi významnejšiu novinku patrí aj audio tunneling do digitálneho signálového procesoru, čo zabezpečí, že sa procesor nepoužíva tak často a tým sa šetrí batéria. Podľa niektorých zdrojov by mala byť výdrž batérie pri prehrávaní vyššia až o 200%. V novom systéme sa tiež objavuje nový experimentálny virtuálny stroj s názvom ART. Jedná sa o kompletne nový virtuálny stroj, ktorý je pravdepodobne plánovaný ako náhrada za doteraz používaný Dalvik. V tejto chvíli sa jedná iba o experimentálnu verziu, ktorá je určená hlavne pre vývojárov. ART používa rozdielny mechanizmus ako Dalvik a vďaka tomu by malo byť spúšťanie aplikácií rýchlejšie. Oproti Dalviku má ART však aj drobnú nevýhodu a tou je zvýšená veľkosť aplikácií. Google zatiaľ nedoporučuje, aby sa ART stal predvoleným resp. základným virtuálnym strojom, pretože by mohlo prísť k vážnym problémom s Androidom. Informácie boli nájdené na [1].

V grafoch nižšie je zobrazené rozdelenie jednotlivých verzií Androidu v dobe písania tejto práce. Z grafov je zrejmé, že viac ako polovica zariadení s Androidom beží na verzii 4.0 a vyššie a predpokladá sa, že vďaka novej optimalizovanej verzii sa fragmentácia Androidu aspoň čiastočne zníži.

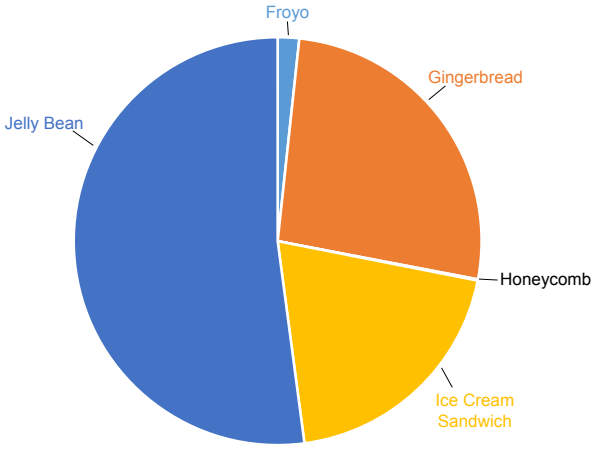
Graf 3.2, ktorý bol prevzatý z [1] zobrazuje prehľadnou formou rozloženie jednotlivých verzií, zatiaľ čo tabuľka ukazuje presné percento týchto verzií.

3.2 Software Development Kit

Sada Android SDK (Software Development Kit) poskytuje potrebné nástroje pre tvorbu, testovanie a ladenie Android aplikácií. Táto sada sa skladá z dvoch častí, ktorými sú zá-

Verzia	Kódové označenie	API level	Zastúpenie
2.2	Froyo	8	1.7 %
2.3.3 - 2.3.7	Gingerbread	10	26.3 %
3.2	Honeycomb	13	0.1 %
4.0.3 - 4.0.4	Ice cream Sandwich	15	19.8 %
4.1.x	Jelly Bean	16	37.3 %
4.2.x		17	12.5 %
4.3		18	2.3 %

Tabulka 3.1: Rozloženie verzii ku dňu 1.11.2013



Obrázek 3.2: Graf rozloženia jednotlivých verzií

kladné nástroje a komponenty špecifické pre konkrétne verzie. Na oficiálnych stránkach je k dispozícii ADT čo je skratka pre Android Developer Tools. ADT sa skladá z Android SDK komponent a vývojového prostredia Eclipse. Tento balíček je dostupný či už pre Linux, Windows alebo Mac. Okrem Vývojového prostredia Eclipse s potrebnými doplnkami (pluginmi), ktoré je zatiaľ preferované prostredie pre tvorbu aplikácii, majú vývojári k dispozícii aj začínajúci projekt s názvom Android Studio. Android Studio je založené na IntelliJ IDEA a je tvorené špeciálne pre vývoj pre Android. Rovnako ako Ecplipse s ADT pluginom aj Android studio poskytuje integrované nástroje potrebné pre vývoj, testovanie a ladenie aplikácii. Android studio je multiplatformé vývojové prostredie dostupné pre Windows, Mac OS X či Linux.

Informácie o Android SDK pochádzajú z [1], [4] a o jazyku JAVA z [6].

Okrem SDK, ktoré je primárne určené pre vývoj v jazyku Java, Google poskytuje pre vývojárov aj Android NDK. Android NDK (Native Development Kit) je súbor nástrojov, ktoré umožňujú implementovanie častí aplikácie pomocou jazykov ako sú C a C++. Pre určitý typ aplikácií je použitie týchto nástrojov lepšia voľba. Typické využitie je napríklad pri CPU-intenzívnych operáciách, ktoré nealokujú príliš veľa pamäte ako napríklad spracovanie signálov, fyzické simulácie a ďalšie. Použitie NDK je vhodné iba v niektorých opodstatnených prípadoch a nie iba preto, že vývojár píše radšej v C alebo C++. Android NDK môže byť integrované do Eclipse ako aj do Visual Studia.

Android Virtual Device

Medzi vývojové nástroje používané pri vývoji patrí aj emulátor zariadení. Ide o špeciálny softvér, ktorý simuluje Android zariadenia. Tento nástroj je dôležitý z viacerých dôvodov. Nielenže umožňuje vyvíjať aplikácie bez fyzického zariadenia, ale taktiež umožňuje testovanie aplikácie na rôznych konfiguráciách. Tento emulátor dokáže emulovať niekoľko zariadení, pričom každá konfigurácia je uložená vo virtuálnych zariadeniach Android (Android Virtual Device - AVD). Tieto virtuálne zariadenia sú vytvárané pomocou AVD managera. AVD ponúka široké možnosti konfigurácie ako verziu systému, rozlíšenie obrazovky, veľkosť operačnej pamäte, údaje o úložisku, ktoré má emulátor simulovať či niektoré pokročilejšie funkcie, akou je napríklad prítomnosť hardvérovej klávesnice.

Android debug bridge

Android debug bridge (ADB) je nástroj umožňujúci komunikáciu s pripojeným Android zariadením alebo emulátorom. Typicky je ADB dostupný cez príkazový riadok, ale existuje aj mnoho grafických užívateľských prostredí pre ovládanie tohto nástroja. Jedná sa o program typu klient - server, ktorý sa skladá z troch častí.

- Klientská časť, bežiaca na Android zariadení, s ktorou je možná komunikácia pomocou ADB príkazov. Ďalšie Android nástroje ako napríklad ADT plugin alebo DDMS tiež vytvárajú ADB klienta.
- Serverová časť, bežiaca ako proces na pozadí na Android zariadení. Serverová časť riadi komunikáciu medzi klientom a ADB démonom bežiacom na emulátore alebo fyzickom zariadení.
- Démon, časť ktorá je spustená ako proces na pozadí na každom emulátore alebo zariadení.

Po štarte klientskej časti ADB sa klient snaží nájsť bežiaci ADB server. Ak server nie je nájdený, je spustený. Po štarte sa server pripojí k TCP portu 5037 a čaká na príkazy posielané z ADB klientov. Každý klient využíva pre komunikáciu so serverom port 5037.

Android Device monitor

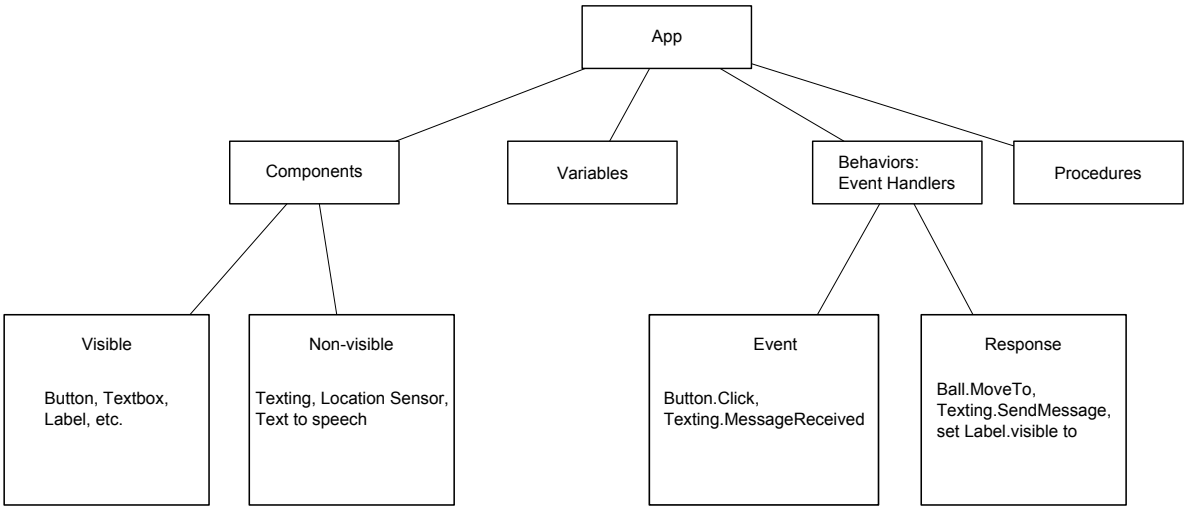
Android Device monitor je samostatný nástroj, ktorý poskytuje grafické užívateľské prostredie pre viaceré aplikácie pre ladenie a analýzu Android zariadení. Android Device manager nevyžaduje žiadnu inštaláciu integrovaného vývojového prostredia ako Eclipse a môže byť použitý ihneď po stiahnutí. Medzi dôležité nástroje patrí napríklad Dalvik Debug Monitor Server (DDMS). Tento nástroj umožňuje zachytenie obrazovky zariadenia, poskytuje informácie o vláknach, procesoch, prichádzajúcich SMS správ či pamäti. Ďalším užitočným nástrojom je Tracer for OpenGL ES. Jedná sa o nástroj pre analýzu OpenGL kódu v Android zariadení. Umožňuje zachytávanie OpenGL príkazov a obrázkov snímok po snímku, pre lepšie pochopenie ako sa jednotlivé príkazy vykonávajú. Za zmienku tiež stojí Hierarchy Viewer, ktorý umožňuje ladenie a optimalizovanie užívateľského prostredia. Poskytuje vizuálny náhľad hierarchie komponentov. Okrem týchto nástrojov obsahuje Device monitor ešte mnoho ďalších.

3.3 Alternatívny vývoj pre platformu Android

Okrem známeho a oficiálne podporovaného jazyka Java s vývojovými prostrediami ako Eclipse alebo Android studio nám Android umožňuje vývoj aplikácií aj v iných programovacích jazykoch alebo frameworkoch. V tejto kapitole postupne predstavím programovanie pomocou MIT App Inventoru, ktorý využíva logické bloky, ďalej sa pozriem na programovanie v jazyku C#, potom spomeniem programovanie s využitím webových technológií ako HTML, CSS a JavaScript a nakoniec pár vetami spomeniem ďalšie, nie tak rozšírené alternatívy.

MIT App Inventor

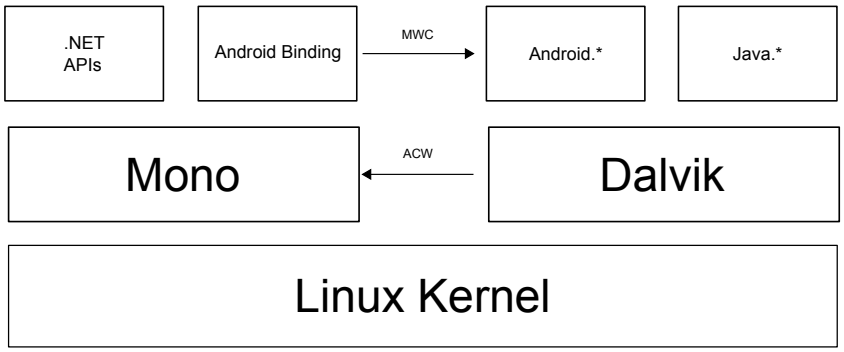
App Invertor je aplikácia pôvodne poskytovaná Googlom, ale momentálne udržiavaná MIT. App Inventor je cloudová služba, ktorá používa grafické rozhranie podobné Scratchu, čo je programovací jazyk určený pre vzdelávanie, kde posúvaním a vytváraním objektov môžme vytvárať Android aplikácie. Je založený na MIT Open Block Java knižnici a poskytuje prístup k funkciám ako GPS, kontakty, webové služby a ďalšie. App Inventor sa skladá z dvoch hlavných častí. Prvá časť je Designer, pomocou ktorého sa tvorí grafické užívateľské prostredie. Druhou časťou je Block Editor, kde sa vytvorí pomocou blokov správanie aplikácie. App Inventor poskytuje širokú škálu typov blokov, ako napríklad kontrolné, logické, textové, matematické a mnoho ďalších. Na obrázku 3.3 je zobrazená architektúra aplikácie vytvorenej pomocou App Inventoru. Tvoria ju 4 základné časti. Jednou z nich sú komponenty. Komponenty sa delia na dva typy a to viditeľné a neviditeľné. Medzi viditeľné patria tie, ktoré sú viditeľné pri spustení aplikácie. Jedná sa napríklad o tlačidlá alebo textové polia. Naopak neviditeľné komponenty nie sú súčasťou používateľského prostredia. Miesto toho poskytujú prístup k vstavaným funkcionalitám zariadenia. Napríklad komponenty LocationSensor alebo TextToSpeech. Ďalším komponentom sú takzvané Event Handlers, ktoré by sa dali preložiť ako zachytávanie udalostí. Tieto udalosti môžu byť buď interné (ako zmena orientácie) alebo externé (ako dáta prichádzajúce z webu). Pomocou nich je implementované chovanie aplikácie tak, že pri objavení udalosti aplikácia reaguje volaním istých funkcií. Funkciou môže byť napríklad zaslanie správy alebo zmena textového poľa.



Obrázek 3.3: Vnútorná architektúra App Inventoru

Xamarin

Firma Xamarin umožňuje vývoj natívnych aplikácii pre Android, iOS alebo Windows Phone pomocou jazykov C# a .NET. Xamarin ponúka dva produkty, MonoTouch a Mono for Android, tiež známe ako Xamarin.iOS a Xamarin.Android. Oba produkty sú postavené nad Mono, open-source verzie .NET frameworku založenom na .NET ECMA štandardoch. Na iOS Xamarin AOT (Ahead-of-Time) kompilátor kompiluje Xamarin.iOS aplikácie priamo do natívneho ARM kódu. Na Androide je aplikácia skompilovaná do IL (Intermediate Language), ktorý je potom skompilovaný JIT (Just-in-Time) do natívneho kódu. V oboch prípadoch využívajú runtime, ktorý sa automaticky stará o veci, ako alokácia pamäte, garbage collection a podobné. Vývoj pomocou Xamarinu je možný v dvoch vývojových prostrediach. Prvý z nich je Visual Studio ku ktorému firma Xamarin vydala add-in, ktorý dopĺňa funkcionalitu potrebnú pre plnohodnotné programovanie či ladenie mobilných aplikácií. Ďalšou možnosť je vývojové prostredie vyvinuté samotnou spoločnosťou - Xamarin Studio. Toto vývojové prostredie ponúka pokročilé funkcie ako refaktorizácia kódu, navigácia, zvýrazňovanie syntaxe, tipy, komentovanie a odsadzovanie, ladenie či vlastný android dizajnér. Podľa oficiálnych stránok je v priemere možné použiť až 75% zdrojového kódu pre medziplatformný vývoj aplikácií. Na obrázku 3.4 je znázornená zjednodušená verzia Xamarin aplikácie pre Android. Preklad Android aplikácie pomocou Xamarinu vytvorí interpretovaný jazyk IL kód. IL kód beží v Mono runtime, ktorý je inštalovaný spolu s mobilnou aplikáciou. Mono runtime beží priamo nad Linux jadrom, nie cez Dalvik. Každý kód, ktorý volá natívne SDK je spúšťaný cez virtuálny stroj Dalvik cez MCW (Managed Callable Wrapper) a ACW (Android Callable Wrapper) komunikačné kanály. ACW je používaný virtuálnym strojom Dalvik pre vrátenie výsledkov späť do Xamarin aplikácie.

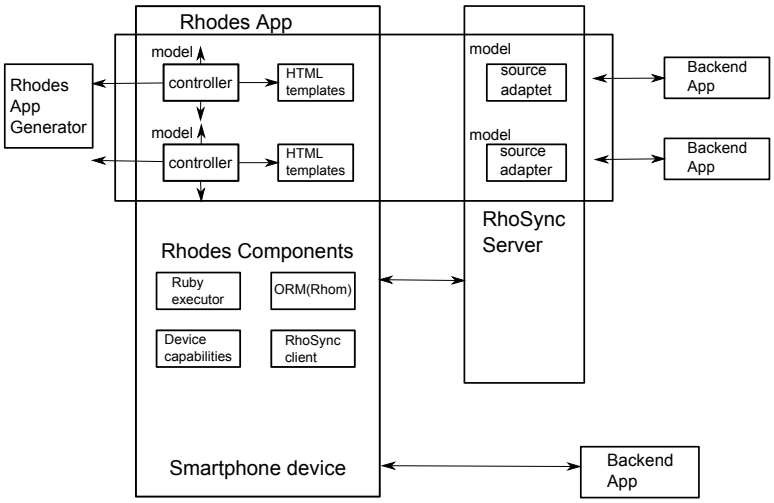


Obrázek 3.4: Schéma architektúry Xamarin

Rhodes

Rhodes je open source framework založený na programovacom jazyku Ruby, ktorý slúži na programovanie natívnych aplikácií (nie webových). Okrem jazyka Ruby využíva Rhodes aj webové jazyky ako HTML, CSS a JavaScript. Rhodes je určený pre vývojárov, ktorí už majú skúsenosti s webovými službami a chcú tvoriť mobilné aplikácie bez nutnosti učiť sa pracovať s SDK a jazykmi pre každú platformu. Rhomobile nástroje a framework umožňujú vývoj či už v prostredí Windows, Linux alebo Mac, avšak vývoj pre špecifické zariadenia vyžaduje nainštalované SDK. BlackBerry a Windows mobile vyžadujú operačný systém Windows, Iphone zariadenia vyžadujú Mac, Android a Symbian bežiace na Jave sú medzi platformové. Rhodes je primárne cielený pre podnikové aplikácie a nie je určený pre vývoj hier alebo

ostatných graficky náročných aplikácii. Vývoj pomocou Rhodes mobile je možný aj bez inštalovania SDK a to vďaka cloudovému vývojovému prostrediu zvanému RhoHub. Okrem cloudového prostredia existuje aj štandardné prostredie nazývané RhoStudio. Obrázok 3.5 zobrazuje architektúru Rhodes Mobile systému. Architektúra zobrazuje jednotlivé časti Rhodes systému a prepojenia medzi nimi.



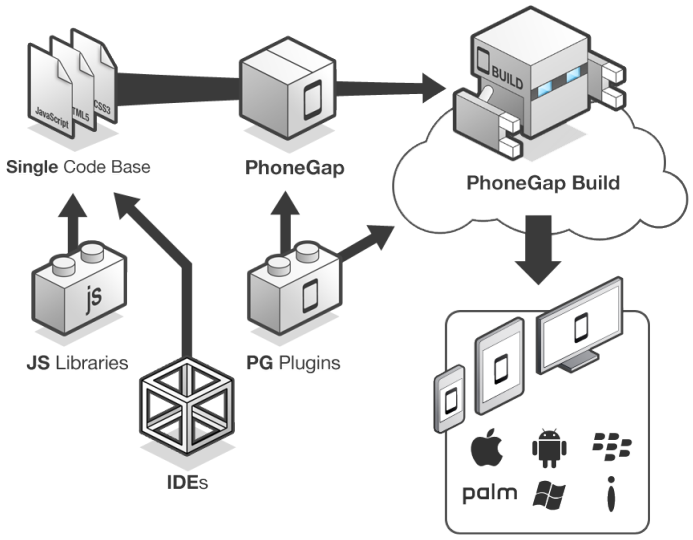
Obrázek 3.5: Schéma architektúry Rhodes

PhoneGap

PhoneGap patrí k open source frameworkom pre tvorbu mobilných aplikácií a podobne ako Rhodes využíva jazyky ako HTML5, CSS3 a JavaScript, ktorý je potreba ovládať na vysokej úrovni. Výsledná aplikácia je tzv. hybrid, to znamená, že kombinuje prvky natívnej a webovej aplikácie. Podporované sú platformy iPhone, Android, BlackBerry , Symbian, Bada alebo WebOS. Tento framework je výborným nástrojom pre vývojárov webových aplikácií, ktoré sa dajú pomocou PhoneGapu zmeniť na mobilnú aplikáciu. Natívne aplikácie majú prístup k určitým funkciám, ktoré nie sú dostupné webovým aplikáciam, ako napríklad kontakty, fotoaparát a ďalšie. PhoneGap toto umožňuje pomocou JavaScript API. Výhodou tohto frameworku je, že nie je nutné sa učiť pracovať s novým SDK každého výrobcu, pretože PhoneGap to spraví za vás. Menšou nevýhodou môže byť pomalosť aplikácií oproti natívnym aplikáciám s rovnakou funkcionalitou. Na obrázku 3.5 je znázornená schéma PhoneGap builderu. Táto schéma zjednodušene zobrazuje, aký je postup pri vytváraní aplikácií pomocou PhoneGap builderu. Detailná práca PhoneGap builderu nie je v tomto prípade popísaná. Ako je možné vidieť na obrázku, PhoneGap je schopný vytvárať aplikácie pre širokú škálu operačných systémov.

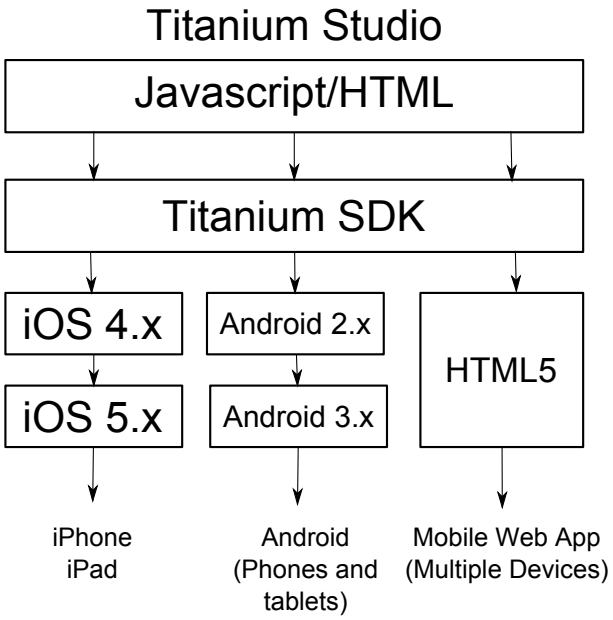
Titanium Mobile

Ďalším frameworkom určeným pre vývoj mobilných aplikácií je Appcelerator Titanium alebo aj Titanium Mobile. Tento framework opäť využíva technológie ako HTML, CSS a JavaScript, no hlavný dôraz je kladený na JavaScript. Titanium Mobile sa zameriava predovšetkým na platformy Android a iPhone. Je dostupný pre Mac, Windows aj Linux. K vývoju pre iPhone je treba Mac a nainštalovaný iPhone SDK. Pre vývoj Android aplikácii



Obrázek 3.6: Schéma práce PhoneGap builderu

je vyžadované Android SDK, ktoré je však možné používať či už na Mac, Windows alebo na Linuxe. Titanium API predstavuje platformovo nezávislý prístup k UI komponentom, ktorými sú napríklad navigačný bar, menu, upozornenia, ale aj súborový systém, zvuky alebo sieť. Obrázok 3.7 zobrazuje architektúru Titanium Mobile, na ktorej môžeme vidieť zjednodušenú ukážku fungovania Titanium Mobile. Aplikácia je písaná prevažne v jazyku JavaScript, ktorý je pomocou Titanium SDK pretvorený do zvoleného formátu pre vybraný operačný systém.



Obrázek 3.7: Architektúra Titanium Mobile

Ostatné

Okrem vyššie spomínaných existujú aj menej známe a menej používané frameworky či programovacie jazyky. Niektoré z nich sú napríklad Corona SDK, ktorá umožňuje vývoj mobilných aplikácií v jazyku LUA, ďalej sú to programovacie jazyky Scala alebo programovací jazyk Mirah, ktorý je založený na jazyku Ruby a jeho syntaxe. Z ďalších môžeme spomenúť napríklad Ruboto, čo je framework umožňujúci vývoj v jazyku Ruby, či jazyk Clojure, ktorý je dialektom programovacieho jazyka Lisp. Do rodiny jazykov Lisp patrí taktiež Kawa framework implementujúci jazyk Scheme. Za spomenutie stojí aj Kivy, čo je open source Python knižnica umožňujúca vývoj aplikácií v jazyku Python. Využitie vyššie spomínaných frameworkov nie je však také časté a výkonnosť týchto aplikácii nemusí byť vždy ideálna, avšak je to jedna z možností alternatívneho vývoja pre platformu Android.

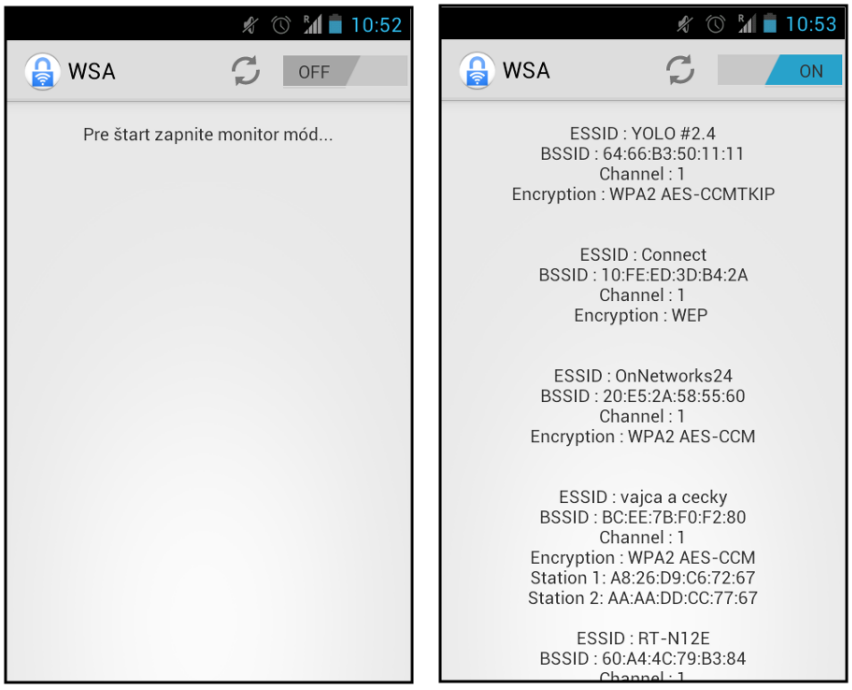
Informácie sú dohľadatelné hlavne v literatúre [8].

Kapitola 4

Návrh aplikácie

Cieľom tejto práce je vytvoriť prototyp aplikácie pre platformu Android. Táto aplikácia bude skenovať bezdrôtové siete v dosahu a analyzovať ich bezpečnosť. Aby aplikácia fungovala je potrebné mať root práva a podporovaný model zariadenia. V tejto kapitole je popísaný návrh aplikácie, či už z hľadiska dizajnu alebo z hľadiska funkčnosti. Taktiež budú popísané zmeny v návrhu a vysvetlenie, prečo k týmto zmenám došlo.

Grafické prostredie



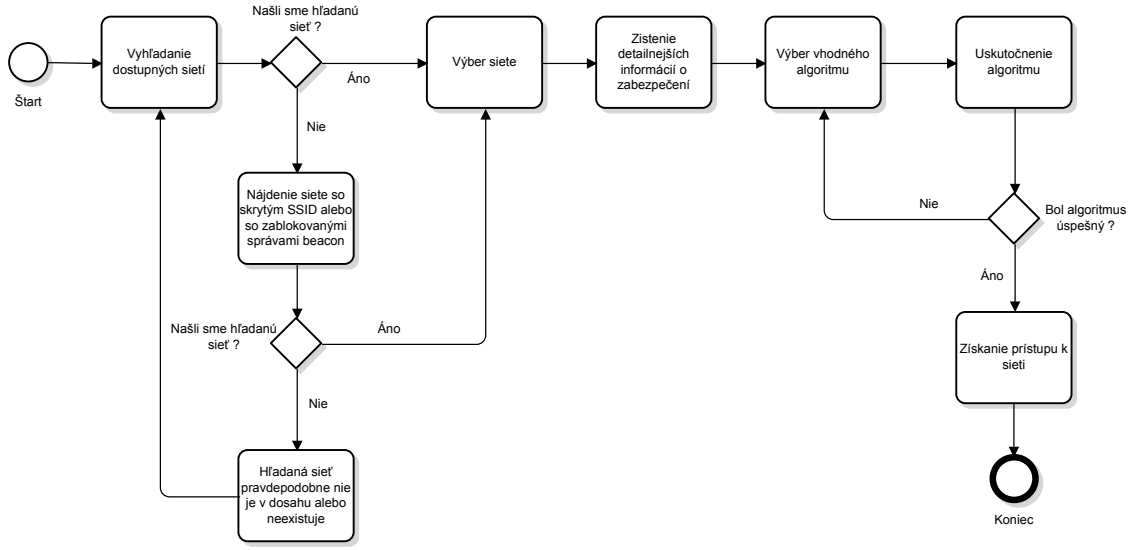
Obrázek 4.1: Návrh grafického užívateľského prostredia

Na obrázku 4.1 je možné vidieť návrh grafického užívateľského prostredia. Aplikácia sa skladá z hlavnej obrazovky, kde sa po zapnutí promiskuitného módu zobrazia dostupné siete v dosahu. Ku každej sieti potom budú dostupné nasledujúce informácie ako BSSID, ESSID, kanál, zabezpečenie a ak je to dostupné, tak aj stanice pridružené k danému prístupovému

bodu. Po kliknutí na vybranú sieť sa aplikácia pokúsi spraviť analýzu tejto siete. Aplikácia ako taká sa skladá z viacerých aktivít. Hlavná aktivita zobrazuje dostupné siete. Ďalšia aktivita patrí rozhraniu pre nastavenia. Sekcia nastavenia momentálne neponúka veľa nastavení, no v budúcnosti by sa to mohlo zmeniť a pridané by mohli byť možnosti výberu útoku či možnosť uloženia výsledných dát. Ako posledná je aktivita zobrazujúca stručné informácie o aplikácii.

Funkčnosť

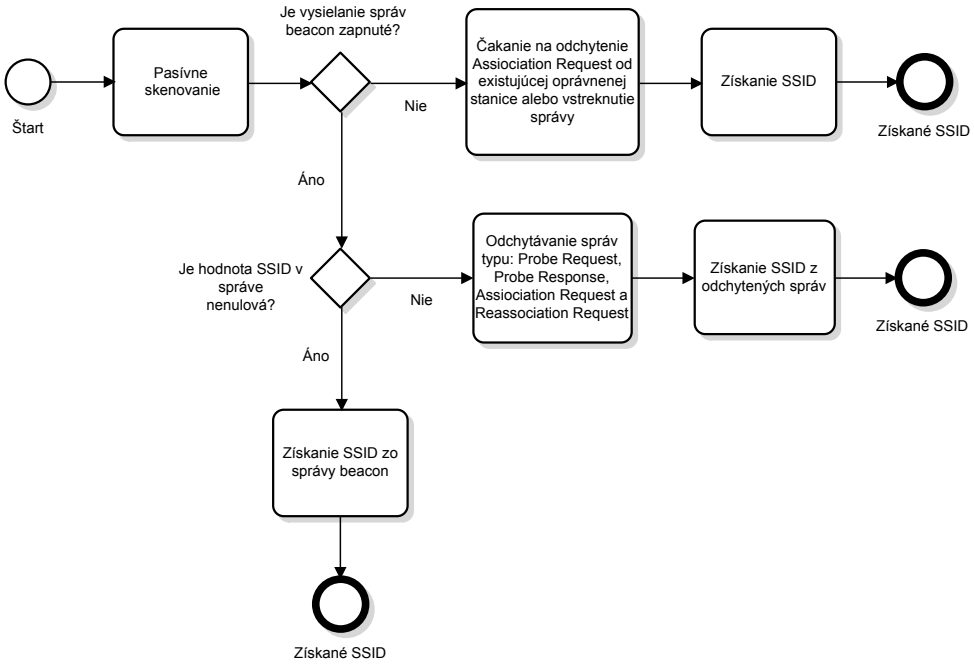
Na obrázku 4.2 je znázornený postup od zapnutia aplikácie cez vyhľadanie siete až po jej prelomenie. Zo začiatku som sa rozhodoval medzi dvoma možnosťami. Pri zapnutí aplikácie by sa automaticky zapol modul s promiskuitným módom a tým by bola aplikácia schopná skenovať dostupné siete. Po prvotných testoch som sa rozhodol toto automatické zapínanie zmeniť a to hlavne z dôvodu vysokých nárokov na batériu, ktoré sú pravdepodobne spôsobené zapnutím promiskuitného módu. Obrázok tiež ukazuje plány, ako zistiť SSID prístupového bodu, ktorý nevysiela správy beacon. Táto funkcionalita bola nakoniec z aplikácie zatiaľ vyradená z dôvodu zložitej implementácie, zhoršenia prehľadnosti aplikácie a taktiež skenovania, ktoré by mohlo trvať príliš dlho. Aplikácia sa zameriava hlavne na prelomenie zabezpečenia typu WEP a to z dôvodu slabšieho výpočtového výkonu, ktorý by bol potrebný pri slovníkových útokoch na WPA. Nie je vylúčené, že pri súčasných trendoch budú mobilné zariadenia v blízkej budúcnosti dostatočne výkonné na to, aby boli schopné efektívne prelomiť aj zabezpečenia typu WPA.



Obrázek 4.2: Návrh postupu analýzy

Ná obrázku 4.3 je návrh algoritmu pre zistenie SSID. Pre zistenie SSID potrebujeme najprv pasívne sledovať sieť, čakať a odchytávať potrebné rámce, v ktorých sa vyskytuje SSID. V normálnom režime prístupový bod vysiela správy beacon, v ktorých sa nachádza SSID. Ak prístupový bod beacon správy nevysiela, sú k dispozícii dva prístupy. Aktívny a pasívny prístup. V prípade pasívneho prístupu prebieha skenovanie a čaká sa na odchytenie Association Request. Naopak, pri aktívnom prístupe sa aplikuje vstreknutie správy. Ak prístupový bod vysiela (toto je defaultné nastavenie) správ beacon, záleží na obsahu týchto správ. Aj tu existujú dve možnosti. Pri prvej možnosti je hodnota SSID nulová a jedná

sa o tzv. skrývanie SSID. V tomto prípade prebieha opäť čakanie a odchytávanie rámcov, ako Probe Request, Probe Response, Assotiation Request alebo Reassotiation Request. Posledný prípad je ten najjednoduchší, čo znamená zapnuté vysielanie beacon správ a nenulová hodnota SSID v týchto správach. V tomto prípade stačí danú správu prijať a jednoducho zistiť dané SSID.



Obrázek 4.3: Návrh algoritmu zistenia skrytého SSID

Kapitola 5

Implementácia a testovanie

Táto kapitola popisuje spôsob implementácie navrhnutej aplikácie. Budú v nej spomenuté využité knižnice, metódy či problémy spojené s implementáciou. Okrem samotnej implementácie tu bude spomenuté aj testovanie aplikácie, ktoré bolo značne obmedzené hlavne kvôli nedostatku zariadení s možnosťou prepnutia sieťovej karty do promiskuitného módu. Vývoj, ladenie a aj testovanie aplikácie prebiehalo na rootnutom zariadení Nexus S.

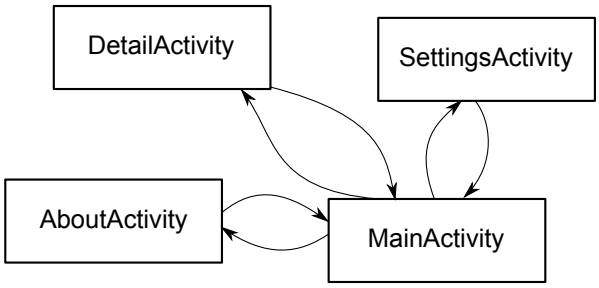
Implementácia

Pre písanie kódu aplikácie je možné použiť mnoho textových editorov. Začiatky prebiehali v oficiálne podporovanom vývojom prostredí Eclipse, o ktorom je písané vyššie. Z dôvodu efektívnejšej práce, rýchlejšej odozve a lepšej podpore nových zariadení som postupne prešiel na beta verziu vývojového prostredia Android Studio. Aplikácia je prakticky rozdelená na dve časti. Časť GUI, ktorá je písaná v jazyku XML a kódová časť písaná v jazyku Java. Okrem tejto možnosti existuje aj možnosť písania celej aplikácie v jazyku Java, ale mnou zvolená možnosť má výhody v prehľadnosti, ľahšom spravovaní a flexibilite.

Hlavná časť grafického užívateľského prostredia je popísaná v súbore **activity_main.xml**, kde je pomocou jazyka xml vytvorený komponent **ListView** slúžiaci na zobrazenie dostupných sietí vo forme zoznamu. Okrem tohto súboru aplikácia obsahuje samostatné xml súbory pre nastavenia **activity_settings.xml** či xml súbor s informáciami o aplikácii **activity_about.xml**. Ďalšie dôležité súbory sú **strings.xml**, ktoré obsahujú textové reťazce či **styles.xml** pre uloženie rôznych štýlov. Za zmienku stojí aj priečinok menu, v ktorom sa nachádzajú súbory pre prácu s tzv. **Action Bar**. Medzi podstatné súbory patrí aj **AndroidManifestl.xml**, v ktorom sú špecifikované práva, ktoré aplikácia využíva, ktorá aktivita je hlavná či aké periférie bude aplikácia používať. V tomto súbore sa taktiež nachádzajú informácie o minimálnej a maximálnej podporovanej verzii Androidu, v ktorej je aplikácia schopná pracovať. Aplikácia tiež využíva jednu externú knižnicu, konkrétne sa jedná o knižnicu RootTools. Táto knižnica bola použitá hlavne pri overovaní root práv a taktiež pri vykonávaní root príkazov v termináli. Tieto príkazy boli využité napríklad pri vytváraní vlastného priečinku, v ktorom sú uložené dočasné súbory. Overovanie root práv sa dá pomocou RootTools zistiť zavolaním jednoduchých metód **isRootAvailable()**, ktorá zisťuje dostupnosť root práv na zariadení a metóda **isAccessGiven()** zase zisťuje, či sú tieto práva povolené pre danú aplikáciu.

Obrázok 5.1 zobrazuje zjednodušenú štruktúru aktivít v aplikácii. Z hlavnej obrazovky, ktorá predstavuje aktivitu **MainActivity** sa dá pomocou menu prejsť do nastavení alebo k zobrazeniu informácií o aplikácii. Nastavenia predstavuje aktivita **SettingsActivity** a

informácie sú zobrazené v aktivite **AboutActivity**. Po výbere konkrétnej siete sa zase dostaneme do aktivity **DetailActivity**, kde sa zobrazujú informácie o vybranej sieti.



Obrázek 5.1: Štruktúra aplikácie rozdelená do aktivít

Pri štarte aplikácie sa zobrazí hlavná obrazovka, ktorá je prázdna a pre začiatok práce s aplikáciou je treba zapnúť promiskuitný mód. Po prepnutí spínača do polohy On sa načíta a následne zapne modul schopný prepnúť sieťovú kartu do promiskuitného módu. V tomto okamihu sa začínajú automaticky prijímať správy beacon od dostupných prístupových bodov. Tieto údaje sú následne ukladané do dočasného xml súboru. Skenovanie sietí prebieha z dôvodu šetrenia batérie a výkonu približne 7 sekúnd. Za tento čas je prijatých väčšina paketov potrebných pre detekciu sietí. Nastavenie tejto premennej by mohlo byť v budúcnosti voliteľné. Pre opätovné skenovanie je užívateľovi dostupné tlačidlo refresh, ktoré opäť spustí skenovanie dostupných bezdrôtových sietí. V správach beacon je obsiahnutých mnoho informácií, ale z dôvodu prehľadnosti sú na základnej obrazovke zobrazené iba niektoré z týchto informácií, ako BSSID, ESSID, úroveň zabezpečenia, číslo kanálu či pridružené stanice.

Po zvolení siete, ktorú chceme analyzovať, prejde aplikácia do aktivity DetailActivity a zobrazí sa obrazovka s informáciami o sieti, ktorú chceme analyzovať. Následne začne aplikácia zbierať pakety z kanálu na ktorom sa zvolená sieť nachádza. Toto zbieranie paketov bude trvať určitý čas. Aby sa urýchlilo zbieranie potrebných paketov, pokúsi sa aplikácia vstreknúť pakety do nami zvolenej siete a tým zvýšiť prevádzku siete. Tieto pakety sú zároveň ukladané do dočasného súboru. Po skončení skenovania sa spustí algoritmus, ktorý sa pokúsi zo získaných dát pomocou PWT útoku prelomiť a získať WEP kľúč. O priebehu postupu aplikácia informuje či už pomocou výpisov na displeji alebo pomocou zobrazených dialógov. Pre úspešné prelomenie kľúča je potrebné nazbierať čo najviac inicializačných vektorov IV, toto číslo je však variabilné a závisí na dĺžke či sile kľúča.

Testovanie

Testovanie aplikácie bolo nakoniec zložitejšie ako pôvodný predpoklad a to hlavne kvôli nedostatku testovacích zariadení. Dôvodom je skutočnosť, že promiskuitný mód na Android zariadeniach nie je také jednoduché spustiť, ako by sa mohlo zdať. Aj keď Android beží na linuxovom jadre a zapnutie/vypnutie promiskuitného modulu by malo byť otázkou okamžiku, opak je však pravdou. Dôvodom je nedostupnosť ovládačov pre sieťové karty na Android zariadeniach. Iba niekoľko zariadení (ako napríklad nexus 7 či nexus one) je zatiaľ schopných využívať promiskuitný mód vďaka dostupným ovládačom a zapnutie tohto módu vyžaduje stiahnutie a nainštalovanie aplikácie bcmon,[1] ktorá tento mód, okrem iného, spúšťa. Ďalší faktor pre spustenie promiskuitného módu je nutnosť povolených tzv. root práv. Tieto limitácie značne obmedzujú možnosti testovania.

[1] http://bcmon.blogspot.com/

Po počiatočných problémoch sa mi podarilo nájsť ovládače a ďalšie potrebné nástroje a spustiť promiskuitný mód na svojom zariadení vďaka zásahom do systému. Pre testovanie bolo použité zariadenie Nexus S s root právami, na ktorý bola nahraná staršia verzia upravenej custom rom s názvom CyanogenMod 9.1 (Android 4.0.4) a ďalej upravená verzia jadra a modul s promiskuitným módom. Telefón disponuje jedno jadrovým procesorom s taktom 1 GHz a 512 MB operačnej pamäte RAM. Aplikácia bola priebežne testovaná počas vývoja. Nakoľko sú potrebné špecifické vlastnosti a nástroje, ktoré sa veľmi ťažko simulujú, bol Nexus S jediným testovaným zariadením. Avšak pri ladení a testovaní bolo využitých čo najviac dostupných nástrojov ako vyššie spomínané DDMS či Traceview.

Možnosti rozšírenia

Ako budúce rozšírenie je možnosť aplikovania viacerých algoritmov schopných prelomiť aj novšie protokoly. Množstvo z týchto algoritmov však využíva nástroje, ktoré v Android zariadeniach nie sú dostupné prípadne sú dostupné v obmedzenej forme, ako napríklad rýchlosť vstrekovania paketov. Medzi ďalšie možnosti patrí viac nastavení , ktoré však môžu byť na úkor prehľadnosti. Do úvahy pripadá aj možnosť rozdelenia aplikácie do dvoch rôznych módov ako napr. základný a rozšírený, kde v rozšírenom móde by bolo dostupných viacero možností. Ďalšou z možností je tiež export do rôznych formátov, ktoré by mohli byť ďalej analyzované.

Kapitola 6

Záver

Cieľom práce bolo navrhnúť a implementovať jednoduchú aplikáciu, ktorá by bola schopná využívať rastúci výpočtový výkon mobilných zariadení a flexibilitu týchto zariadení k analýze bezpečnosti bezdrôtových sietí. Pre dosiahnutie týchto cieľov bolo potrebné využitie promiskuitného módu sieťovej karty mobilného zariadenia. Problémy spojené so zapnutím tohto módu sú popísané v kapitole 5.

Po naštudovaní problematiky a teórie o bezpečnosti bezdrôtových sietí, ktorá je popísaná v kapitole 2 a programovania pre platformu Android (kapitola 3) bola v ďalšej kapitole 4 navrhnutá aplikácia so základnou funkcionalitou zameranou hlavne na analýzu sieti s protokolom WEP, ktorý je spomedzi popísaných protokolov najviac zraniteľný. V tejto kapitole je tiež popísaný postup a využité prostriedky pri implementácii aplikácie. Aplikácia prijíma správy beacon od prístupových bodov v okolí a zobrazuje siete v prehľadnom formáte. Po výbere siete prevedie aplikácia radu algoritmov pre analýzu a informuje o dosiahnutom výsledku. Implementovaná aplikácia dokáže úspešne prelomiť jednoduchý WEP kľúč po nazbieraní dostatočného množstvo inicializačných vektorov. Potrebné množstvo inicializačných vektorov IV závisí na dĺžke a sile zisťovaného kľúča.

Na základe získaných skúseností a testovania sa dá zhodnotiť, že aj keď je tu určitá možnosť testovania bezpečnosti bezdrôtových sietí pomocou mobilného zariadenia, tak je dosť obmedzená. Obmedzenia sú spôsobné hlavne nedostupnosťou ovládačov či v niektorých prípadoch slabšiemu výpočtovému výkonu. Ako príklad rastúceho záujmu v tej oblasti bol spomenutý PwnPhone v podkapitole 2.7, ktorá niektoré z problémov rieši pridaním externého príslušenstva a upraveného softvéru. Výsledná aplikácia bola vyvíjaná aj testovaná na mobilnom zariadení Nexus S, ktorý mal povolené root práva a okrem upravenej verzie Androidu obsahoval aj modul schopný zapnúť promiskuitný mód. Z tohoto dôvodu aplikácia nebola nahraná do Google Play obchodu, no nie je vylúčené, že sa táto situácia v budúcnosti zmení. Posledná podkapitola zhŕňa niektoré nápady na vylepšenia či možné úpravy v budúcnosti.

Literatura

[1] Android dokumentácia [online]: `http://www.developer.android.com`.

[2] Cyrus Peikari: *Maximum Wireless Security*. Sams Publishing, 203, iSBN 978-0-672-32488-1.

[3] Erik Tews, Martin Beck: *Practical attacks againts WEP and WPA*. ACM, 2009, iSBN 978-1-60558-460-7.

[4] Grant Allen: *Android 4*. Computer Press Brno, 2013, iSBN 978-80-251-3782-6.

[5] Hakima Chaouchi, Maryline Laurent-Maknavicius: *Wireless and Mobile Networks Security*. Wiley, 2009, iSBN 978-1-84821-117-9.

[6] Jeff Friesen: *Learn Java for Android Development, 2nd Edition*. Apress, 2013, iSBN 978-1-4302-5722-6.

[7] Johny Cache, Joshua Wright, Vincent Liu: *Hacking Exposed Wireless, 2nd Edition*. McGraw-Hill Osborne Media, 2010, iSBN 978-0-0716-6661-9.

[8] Lauren Darcey, Shade Conder: *Pro Smartphone Cross-Platform Development: iPhone, Blackberry, Windows Mobile and Android development and Distribution*. Apress, 2010, iSBN 978-1-4302-2868-4.

[9] Lee Barken: *Jak zabezpečit bezdrátovou síť Wi-Fi*. Computer Press Brno, 2004, iSBN 80-251-0346-3.

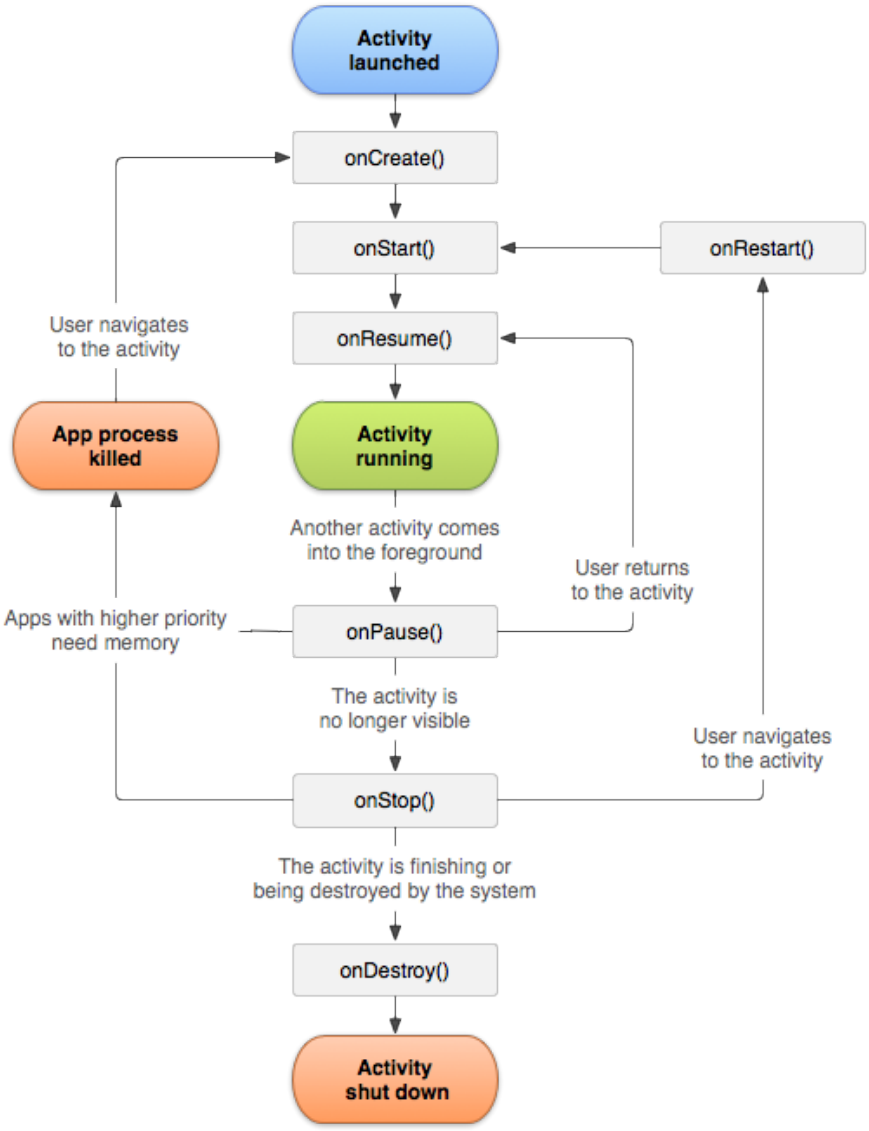
[10] Reto Meier: *Professional Android Application Development*. Wiley, 2009, iSBN 978-0-470-34471-2.

[11] Rita Pužmanová: *Bezpečnost bezdrátové komunikace*. Computer Press Brno, 2005, iSBN 80-251-0791-4.

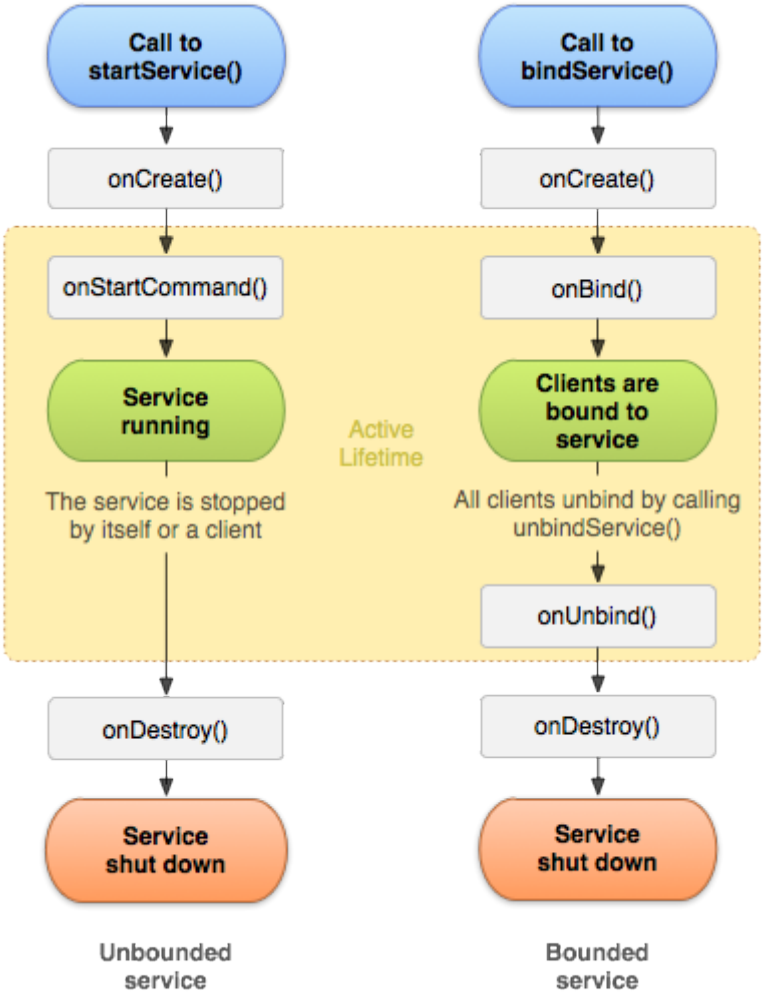
[12] Wolfgang Osterhage: *Wireless Security*. Science Publishers, 2011, iSBN 978-1-57808-768-6.

Příloha A

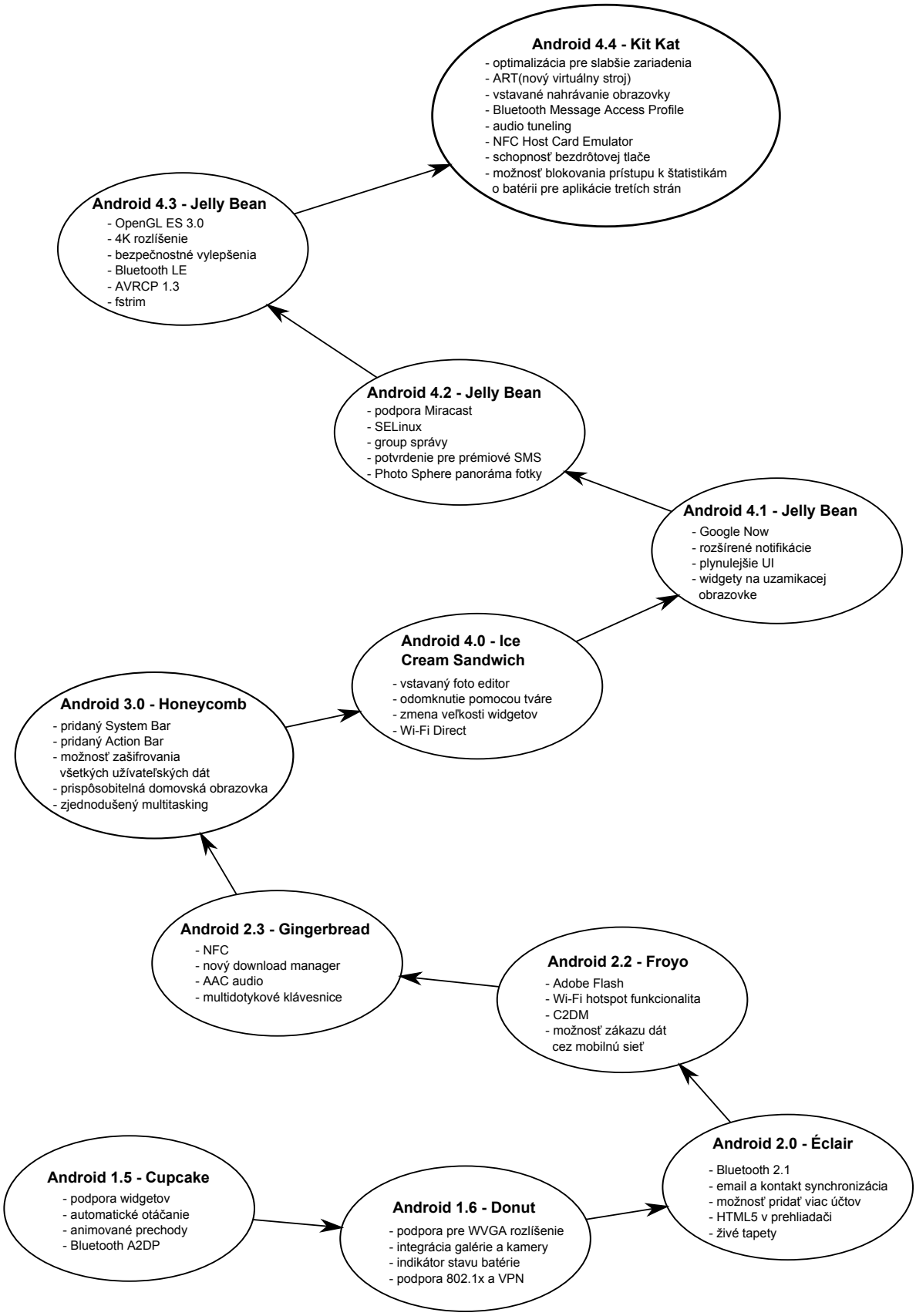
Obrázky



Obrázek A.1: Životný cyklus aplikácie



Obrázek A.2: Životný cyklus služby



Obrázek A.3: Infografika jednotlivých verzií Androidu

Příloha B

Obsah CD

- zdrojové súbory mobilnej aplikácie
- inštalačný súbor mobilnej aplikácie (APK)
- zdrojové súbory tejto práce
- práca vo formáte pdf
- custom rom použitá pre správny chod aplikácie vo forme .zip súboru
- .zip súbor s upraveným jadrom, ovládačmi a monitor mode modulom
- README súbor
- video s ukážkou ako program pracuje