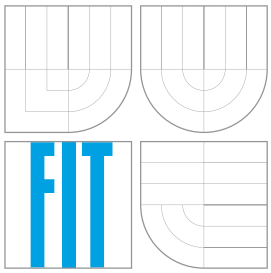
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

MOBILNÍ APLIKACE PRO TESTOVÁNÍ ZRANITELNOSTÍ DNS

MOBILE APPLICATION FOR DNS VULNERABILITIES TESTING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Ing. MICHAL BÉDER
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL KOVÁČIK
SUPERVISOR

BRNO 2016

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Béder Michal, Ing.**
Obor: Informační technologie
Téma: **Mobilní aplikace pro testování zranitelností DNS**
Mobile Application for DNS Vulnerabilities Testing
Kategorie: Počítačové sítě

Pokyny:

1. Seznamte se se službou a protokolem DNS, nastudujte možnosti a specifika tvorby mobilních aplikací pro Android. Seznamte se s existujícími mobilními aplikacemi podobného charakteru.
2. Analyzujte požadavky na vlastní mobilní aplikaci pro testování zranitelností DNS.
3. Uvedenou aplikaci navrhněte.
4. Implementujte navrženou aplikaci a otestujte ji na reálných sítích.
5. Zhodnoťte výslednou aplikaci, její ovladatelnost a přehlednost. Navrhněte možnosti dalšího rozšíření.

Literatura:

- Dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Bez požadavků.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kováčik Michal, Ing.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Abstrakt

Táto práca rieši spôsob návrhu aplikácie pre systém Android, ktorý umožňuje plnú kontrolu nad vytváraním paketov nesúcich protokol DNS. Toho je dosiahnuté implementáciou časti aplikácie v jazyku C++. Ďalej sú v práci popísané bezpečnostné obmedzenia systému Android a možnosti kombinovania natívneho kódu s Java kódom. Výsledná aplikácia umožňuje generovať vybrané sieťové útoky využívajúce známych slabín systému DNS.

Abstract

The aim of this thesis is to show the way how to implement an Android application, which allows full control over DNS packets creation. This was achieved by partial implementation of the application in C++ programming language. Furthermore, in this thesis are described topics as Android security model and possibilities of combining native and Java code. Resulting application allows to generate network attacks, which are exploiting some of the known DNS vulnerabilities.

Kľúčové slová

DNS, Android, NDK, amplifikácia, DoS, útok, tunelovanie, natívny

Keywords

DNS, Android, NDK, amplification, DoS, attack, tunneling, native

Citácia

BÉDER, Michal. *Mobilní aplikace pro testování zranitelností DNS*. Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Příjmení Jméno.

Mobilní aplikace pro testování zranitelností DNS

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing. Michala Kováčika. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Michal Béder
16. mája 2016

Poďakovanie

Týmto by som chcel poďakovať vedúcemu práce Ing. Michalovi Kováčikovi za ochotu, rady a odborné pripomienky behom konzultácií, ktoré mi pomohli bakalársku prácu vypracovať.

Obsah

Kapitola 1

Úvod

Mobilné zariadenia, akými sú telefóny alebo tablety, dokážu so zvyšujúcim sa výkonom nahradiť bežnému používateľovi tradičné stolové počítače a notebooky. Tento trend je spojený s vyšším komfortom, ktorý tieto zariadenia poskytujú. S rastúcim dopytom vzniká celá škála nového softvéru pre jednotlivé mobilné platformy.

Na poli mobilných operačných systémov dosiahol najväčšieho trhového podielu systém Android, ktorý stále podlieha intenzívnemu vývoju. Vývojárom umožňuje tvorbu aplikácií, ktoré boli donedávna doménou klasických počítačov. Používateľom zase rýchly a jednoduchý prístup k množstvu aplikácií. V tejto práci postupne ukážem akými spôsobmi je možné vyvíjať aplikácie pracujúce na najnižších vrstvách sieťového ISO-OSI modelu.

Domain Name System (ďalej DNS) je jedným zo základných pilierov sieťovej komunikácie. Pre mobilné zariadenia, ktoré sú najviac využívané na prehliadanie webu a posielanie elektronickej pošty je kľúčový. Každé toto zariadenie sa pravidelne dostáva do kontaktu s rôznymi DNS servermi, práve vďaka svojej najväčšej výhode, ktorou je mobilita a možnosť na rôznych miestach sa jednoducho pripájať na internet prostredníctvom Wi-Fi sietí. Tohto faktu využívajú útočníci, ktorí sú tak schopní pri vynaložení rovnakého úsilia zasiahnuť rôznymi typmi útokov väčšie množstvo zariadení.

V poslednom desaťročí je zabezpečenie DNS serverov výrazne lepšie aj vďaka algoritmom detekujúcim rôzne pokusy o útoky. Táto práca ukazuje ako je možné takéto útoky vytvárať prostredníctvom mobilnej aplikácie, ktorá primárne slúži na jednoduché testovanie zabezpečenia DNS serverov a na generovanie dát pre detekčné algoritmy.

Celá problematika je rozdelená na dve hlavné časti, ktorými sú tvorba aplikácií pre operačný systém Android popísaná v kapitole 2 a služba DNS popísaná v kapitole 3. Poznatky z týchto častí sú využité pri návrhu a implementácii aplikácie, ktorú popisujú kapitoly 4 a 5. Aplikácia je implementovaná v jazykoch C++ a Java. Java ako primárny programovací jazyk pre Android je využitá na tvorbu grafického používateľského rozhrania a spúšťanie programov napísaných v natívnom C++ kóde. Tie tvoria jadro aplikácie a poskytujú prístup k tvorbe paketov na najnižších úrovniach.

Kapitola 2

Android

Táto kapitola popisuje mobilný operačný systém Android a vývoj aplikácií na tejto platforme. Spoločne s druhou kapitolou poskytuje teoretický základ nevyhnutný pre správny návrh a implementáciu požadovanej funkcionality aplikácie.

V roku 2005 začala s vývojom operačného systému Android spoločnosť Google. Následne sa behom piatich rokov stal vedúcou mobilnou platformou. Od začiatku sa jedná o „open source“ projekt, aj keď nie úplne typický. To umožnilo vznik modifikovaných verzií založených na Android-e. Príkladmi môžu byť CyanogenMod alebo MIUI. Súčasný podiel jednotlivých verzií na trhu je uvedený v tabuľke 2.1.

API	10	15	16-18	19	21-22	23
Podiel na trhu	2,6 %	2,2 %	21,3 %	33,4 %	35.8 %	4,6 %

Tabuľka 2.1: Podiel jednotlivých API na trhu – údaje z 4.apríla 2016 (prevzaté z [2])

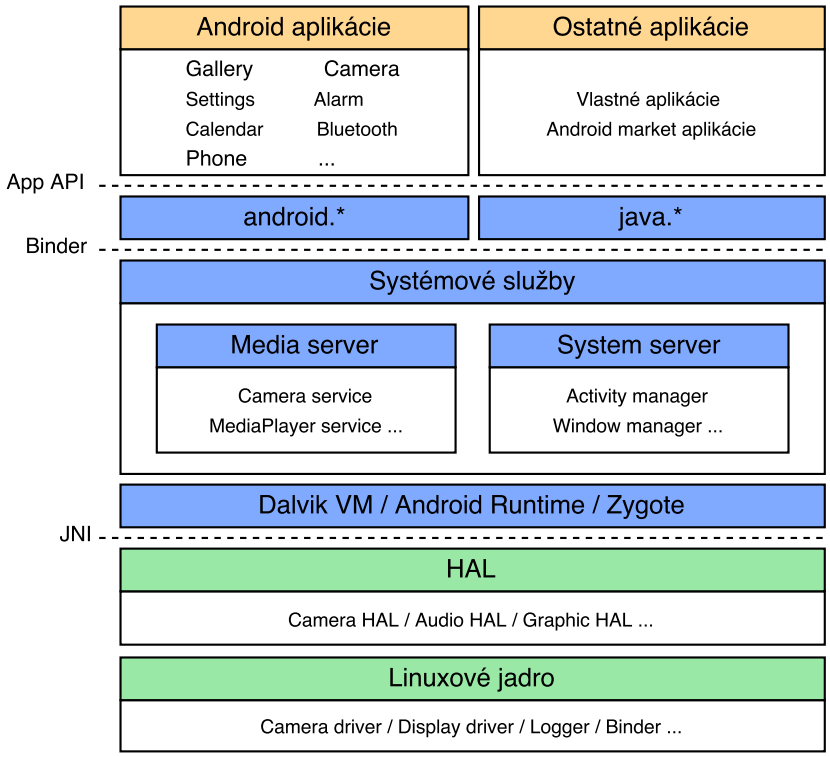
2.1 Architektúra systému Android

Zjednodušený model architektúry[1] určený pre vývojárov aplikácií rozoznáva 5 hlavných komponent. Tými sú od najvrchnejšej vrstvy aplikácie, aplikačné rozhranie, knižnice, Android runtime a linuxové jadro. Vhodné je však spomenúť aj presnejší model, ktorý uvádza ako oficiálna príručka [3], tak aj Yaghmour vo svojej knihe [19]. Tento model je možné vidieť na obrázku 2.1. Nasleduje popis najdôležitejších komponent:

- Linuxové jadro – klasické linuxové jadro, mierne upravené pre potreby mobilných zariadení ako sú úspora energie alebo menšia operačná pamäť.
- HAL (hardware abstraction layer) – definuje štandardné rozhranie pre výrobcov hardvéru, a zároveň umožňuje vyšším vrstvám Android-u nezávislosť na implementácii jednotlivých ovládačov.
- Dalvik virtual machine – upravená verzia „Java Virtual Machine“ pre potreby mobilných zariadení podobne ako tomu bolo pri linuxovom jadre. Jeho úlohou je kompilovať zdrojové Java kódy do bajtkódu a ten za behu interpretovať .

[1] Viz. http://developer.android.com/images/system-architecture.jpg.

- Systémové služby – sprostredkovávajú komunikáciu medzi aplikačným rozhraním a nižšími vrstvami. Umožňujú takto prístup vyšších vrstiev k hardvérovým komponentám.
- Binder IPC – pomocou vzdialeného volania procedúr (RPC) sa stará o medziprocesovú komunikáciu (IPC) medzi aplikačným rozhraním a systémovými službami. Vývojár s ním neprichádza priamo do kontaktu.
- JNI (Java Native Interface) – predstavuje premostenie medzi Java kódom a kódmi napísanými v jazykoch C/C++.
- Aplikačné rozhranie – sprístupňuje vývojárom dostupnú funkcionalitu zapuzdrenú v triedach jazyka Java.



Obr. 2.1: Architektúra systému Android (inšpirované z [19])

Procesy

Viacprocesové prostredie Android-u umožňuje vedľa seba existovať rôznym aplikáciám od rôznych výrobcov. Takisto ako pri unixových systémoch sa pri vytváraní nových procesov používa stratégia Copy-on-write (ďalej COW). To znamená, že nový proces je presná kópia rodičovského procesu, ale pri jeho vzniku sa nekopírujú všetky dáta. Tie sa skopírujú až po tom, čo sa nový proces pokúsi o zápis. Je tak umožnený rýchlejší štart nových procesov.

Každá aplikácia beží ako samostatný proces a systém pre ňu vytvorí novú inštanciu virtuálneho stroja. Bolo by neefektívne pre každý nový proces (aplikáciu) načítavať všetky zdroje a Java triedy zakaždým, keď sa vytvorí nová inštancia virtuálneho stroja. Preto existuje šablónový proces, od ktorého sa oddeľujú (fork) nové procesy aplikácií. Volá sa *Zygote* a beží ako systémový démon, ktorého jedinou úlohou je vytvárať nový proces pre

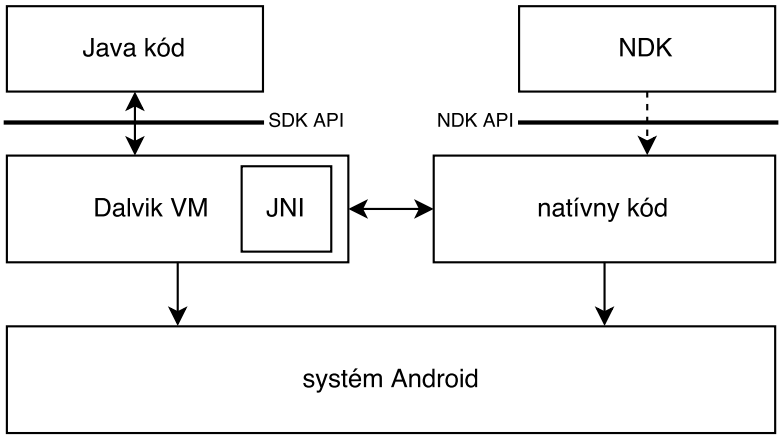
každú aplikáciu. *Zygote* sa takto stáva rodičovským procesom všetkých aplikácií. Pri svojom štarte načíta všetky potrebné Java triedy, zdroje a otvorí socket, na ktorom očakáva žiadosti o štart novej aplikácie. Každá aplikácia tak, po oddelení, získa všetky načítané dáta, ktoré obsahoval *Zygote*. Systémové knižnice, ktoré používa Android nie sú zapisovateľné. To znamená, pri stratégii COW, že každá aplikácia zdieľa rovnakú kópiu Java tried a zdrojov v pamäti ako *Zygote*. Vďaka tejto architektúre sa šetrí operačná pamäť a každá novo spustená aplikácia má výrazne nižší vplyv na množstvo spotrebovanej operačnej pamäte. Tieto informácie sú dostupné v dielach od Mednieksa [12] a Yaghmoura [19].

2.2 NDK

NDK (Native Development Kit) predstavuje súbor nástrojov, ktoré umožňujú napísať časť alebo celú aplikáciu v natívnom C/C++ kóde. Býva často využívaný hernými aplikáciami kvôli vyššiemu výkonu, ktorý však nie je zaručený. Volanie natívneho kódu prostredníctvom JNI prináša so sebou ďalšiu réžiu, preto nemusí byť lepší výkon pravidlom. Ďalšími dvoma výhodami, ktoré NDK poskytuje sú:

- využitie C/C++ knižníc vyvinutých pôvodne pre iné operačné systémy bez nutnosti prepisovania do jazyka Java,
- prístup k nižším vrstvám systému.

V iných prípadoch, než sú uvedené, by sme sa mali používaniu natívneho kódu v aplikáciách vyhýbať. Princíp interakcie natívneho kódu s Java kódom a súvislosti s NDK a SDK[2] sú naznačené na obrázku 2.2. Jedným z mála ucelených diel, ktoré sa zaoberá touto problematikou je monografia [10]. Z tohto diela a NDK dokumentácie [4] čerpá aj táto podkapitola.



Obr. 2.2: Interakcie natívneho a Java kódu (inšpirované z [10])

Zdrojové súbory s natívnym kódom je potrebné umiestniť do adresára `jni/`. V tomto adresári by sa mal nachádzať súbor `Application.mk`. Ten obsahuje informácie platné pre všetky aplikácie, ktorých zdrojové kódy sa nachádzajú v adresárovej štruktúre s koreňovým

[2]SDK (Software Development Kit) – súbor nástrojov umožňujúci vývoj pre platformu Android, obsahuje knižnice, debugger, emulátor, dokumentáciu atď.

adresárom `jni/`. Ďalším potrebným súborom je `Android.mk`. Funguje veľmi podobne ako `makefile` známy z unixového prostredia.

Zariadenia s operačným systémom Android obsahujú rôzne typy procesorov, ktoré podporujú rôzne inštrukčné sady. Na túto skutočnosť treba prihliadať z dôvodu, že jazyky C/C++ sú kompilované a je potrebné poznať cieľovú hardvérovú platformu už v dobe kompilácie. Každá kombinácia procesoru a jeho inštrukčnej sady má svoje aplikačné binárne rozhranie (ABI). Všetky podporované ABI, spolu s nimi podporovanými inštrukčnými sadami, sú zhrnuté v tabuľke 2.2.

ABI	**Podporované inštrukčné sady**
armeabi	ARMV5TE, Thumb-1
armeabi-v7a	armeabi, Thumb-2, VFPv3-D16
arm64-v8a	AArch-64
x86	x86 (IA-32), MMX, SSE/2/3, SSSE3
x86_64	x86-64, MMX, SSE/2/3, SSSE3, SSE4.1, SSE4.2, POPCNT
mips	MIPS32r1
mips64	MIPS64r6

Tabuľka 2.2: Podporované ABI a nimi podporované inštrukčné sady (prevzaté z [4])

Android studio a NDK

Android studio (ďalej *AS*), postavené na platforme *IntelliJ IDEA*, sa od roku 2013 stalo oficiálnym integrovaným vývojovým prostredím podporovaným spoločnosťou Google. Problém s integráciou NDK však ešte pretrváva aj v roku 2016. Historicky sa NDK a SDK distribuovali oddelene. V najnovších verziách *AS* je snaha o zjednotenie týchto nástrojov. Z tejto iniciatívy vzniká aj nový nástroj slúžiaci na automatické zostavenie projektov v *AS* – *Experimental Gradle*, ktorý nahrádza pôvodný spôsob prekladu pomocou súborov `Android.mk`. Prináša so sebou novú DSL[3] syntax a funguje zatiaľ len v spolupráci so špecifickou verziou pôvodného jazyka Gradle. Zoznam verzií a všetky potrebné informácie sú k dispozícii v používateľskej príručke [5]. Dôležité je, že v súčasnosti tento systém nepodporuje preklad zdrojových kódov s vlastným prekladacím systémom. Jeho vývoj však napovedá, že v budúcnosti sa s NDK ráta a jeho využitie v Android aplikáciách bude výrazne jednoduchšie.

2.3 Bezpečnosť

Linuxové jadro je celosvetovo rozšírené a používané mnohými, na bezpečnosť citlivými, systémami. Jeho bezpečnostné prvky sú tak rokmi preverované a opravované množstvom programátorov. K nim Android pridáva vlastné bezpečnostné prvky špecifické pre mobilnú platformu (viz. príručka [6]). Najdôležitejšími z nich sú:

- izolácia procesov,
- systém oprávnení na báze používateľov,

[3]DSL (Domain Specific Language) – Gradle je špeciálny jazyk určený pre automatizáciu, založený na jazyku Groovy viz. http://www.groovy-lang.org/

- zabezpečenie medziprocesovej komunikácie,
- striktné oddeľovanie zdrojov jednotlivých používateľov.

Mobilné zariadenia bývajú, na rozdiel od notebookov a stolových počítačov, spravidla vlastnené jedným používateľom. Tento fakt umožnil vývojárom Android-u zaujímavo využiť linuxovú podporu viacerých používateľov. Systém pridelí každej aplikácii unikátne ID, tým pádom každá aplikácia vystupuje ako samostatný používateľ so svojimi vlastnými právami. Štandardne tak aplikácie nemajú vzájomne prístup k svojím dátam a k operačnému systému len obmedzený. Tento mechanizmus sa nazýva „aplikačný sandbox“ a ako vyplýva z predchádzajúcich viet, funguje na úrovni jadra operačného systému. Podlieha mu všetok softvér nachádzajúci sa vo vyšších vrstvách než jadro (viz. obr. 2.1). To zahŕňa systémové knižnice, aplikačné rozhranie, aplikácie, ale aj natívny kód.

Na štandardnom mobilnom zariadení so systémom Android má práva superpoužívateľa (ďalej `root`) len niekoľko systémových aplikácií. Aplikácie s `root` oprávneniami majú prístup ku všetkým dátam, môžu ľubovoľne upravovať operačný systém aj iné aplikácie. Takýto používateľ má, za cenu zvýšeného bezpečnostného rizika, prístup prakticky ku všetkému v rámci celého systému. Z hľadiska vývojára je takto možné pristupovať k funkciám, ktoré neposkytuje Android API alebo ich za bežných okolností nepovoľuje jadro. Informácie o bezpečnosti sú dostupné z používateľskej príručky Android-u [6]. Táto podkapitola čerpala ďalej aj z monografie [19].

Kapitola 3

DNS

Zatiaľ čo v minulej kapitole boli popísané nástroje potrebné pre implementáciu aplikácie, táto kapitola poskytuje informácie nevyhnutné k pochopeniu činnosti aplikácie.

Systém doménových mien tvorí celosvetovú distribuovanú databázu informácií, nezbytnú pre sieťovú komunikáciu. Časti tejto databázy sú umiestnené na tzv. menných serveroch (nameserver). Jeho primárnou úlohou je mapovanie doménových mien na IP adresy. Dnes však obsahuje podstatne viac informácií, ku ktorým sa dostaneme ďalej v tejto kapitole. Ich prehľadný popis uvádza kniha Síťové aplikace a jejich architektura [11]. Existencia doménových mien je vhodná nielen pre používateľa. Doménové meno nezávisí na konkrétnej IP adrese. Pokiaľ existuje pre jedno doménové meno viac záznamov s rôznymi IP adresami, menný server potom tieto záznamy v odpovediach rovnomerne rotuje.

3.1 Architektúra DNS

Architektúru DNS tvoria tri hlavné komponenty a to:

- Priestor doménových mien – je hierarchicky usporiadaná databáza, ktorá tvorí koreňový strom. Koreňom je uzol `root`. Všetky ostatné uzly sú pomenované textovým reťazcom, ktorý tvorí časť doménového mena. Plné doménové meno (FQDN, Fully Qualified Domain Name) tvorí postupnosť názvov uzlov, až ku koreňu stromu, oddelených bodkami (napr. „www.example.com.“).
- Servery DNS (menné servery) – sú programy, z ktorých každý uchováva v pamäti časť dát priestoru doménových mien. Ako uvádzajú autori Matoušek [11], Kabelová a Dostálek [9], existuje viac typov týchto serverov:
  a) Primárny menný server – poskytuje autoritatívne informácie o doménach, ktoré spravuje.
  b) Sekundárny menný server – získava informácie o doménach od primárneho servera. Pre tieto domény je tiež autoritatívnym serverom.
  c) Záložný menný server – požiadavky preposiela iným serverom a sám si uchováva informácie vo svojej cache pamäti. Poskytuje neautoritatívne informácie.
  d) Koreňový menný server – je špeciálnym typom primárneho menného servera. Obsluhuje doménu `root`.
  e) Tajný (stealth) menný server – autoritatívny server, známy len serverom, ktoré majú jeho adresu staticky nakonfigurovanú.

- Resolvery – sú klientské programy posielajúce požiadavky menným serverom. Každý resolver musí poznať adresu aspoň jedného menného servera.

3.2 Syntax doménových mien DNS

Doménové mená tvoria textové reťazce oddelené bodkami. Prvý reťazec je, v prípade FQDN, označením sieťového rozhrania konkrétneho počítača. Každý ďalší reťazec označuje doménu vyššieho rádu. Maximálna dĺžka reťazca je 63 znakov. Povolené sú všetky alfanumerické znaky a pomlčka. Reťazec musí začínať písmenom a končiť číslom alebo písmenom. Veľké a malé písmená sa nerozlišujú. Celková dĺžka doménového mena nesmie presiahnuť 255 oktetov. V praxi je jednoduchšie uvažovať limit 253 znakov. Je to zapríčinené tým, že každé doménové meno sa prekóduje spôsobom uvedeným v tabuľke 3.1. Prekódované doménové meno sa skladá z čísel reprezentujúcich dĺžku následného reťazca, textových reťazcov a je zakončené číslom 0. V ukážke bolo, na zakódovanie 15tich znakov vrátane bodiek, potrebných 17 oktetov. Analogicky dostaneme z limitu 255 oktetov maximum 253 znakov. Tento postup a celú syntax doménových mien popisuje technická správa RFC 1035 [13].

	w	w	w	.	e	x	a	m	p	l	e	.	c	o	m	
								⇓								
3	w	w	w	7	e	x	a	m	p	l	e	3	c	o	m	0

Tabuľka 3.1: Prekódovanie doménového mena v DNS

3.3 DNS záznamy

Všetky informácie v celom systéme DNS sú uložené v podobe DNS záznamov (resource records) v jednotnom formáte. Tento formát je použitý rovnako v zónových súboroch uložených na DNS serveroch, ako aj v DNS odpovediach (viz. podkapitola 3.4). Popisuje ho RFC 1035 [13] a vyzerá takto:

- NAME – meno uzlu, ktorému daný záznam patrí.
- TYPE – 16 bitová hodnota určujúca typ záznamu.
- CLASS – 16 bitová hodnota určujúca triedu záznamu.
- TTL – 32 bitové znamienkové celé číslo špecifikujúce čas, počas ktorého môže byť DNS záznam uložený v pamäti cache. Po uplynutí tohto času musí menný server príslušný záznam zmazať. Pokiaľ má hodnotu 0 znamená to, že daný záznam sa nesmie do pamäte vôbec ukladať.
- RDLENGTH – 16 bitové bezznamienkové celé číslo, ktoré určuje dĺžku poľa RDATA v bajtoch.
- RDATA – textový reťazec premenlivej dĺžky. Jeho formát závisí od typu a triedy záznamu.

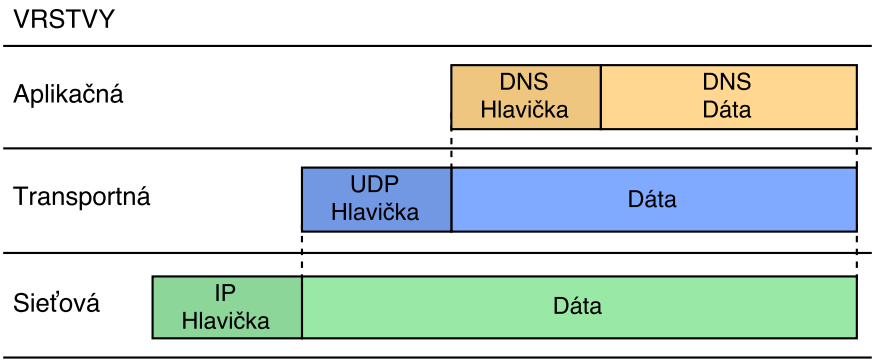
Množiny typov a tried záznamov sa líšia podľa toho, či sa jedná o informácie uložené na mennom serveri alebo o informácie uvedené v DNS požiadavke. Množiny týchto údajov, ktoré môžu byť súčasťou DNS požiadavky, sú obsiahlejšie a v rámci prehľadnosti bývajú označené písmenom „Q“ na začiatku (t.j. QTYPE a QCLASS). Niektoré typy záznamov z množiny QTYPE sú uvedené v tabuľke 3.2.

Typ	Hodnota	Popis
A	1	IPv4 adresa pre dané doménové meno
NS	2	autoritatívny menný server pre danú doménu
CNAME	5	kanonické meno pre daný alias doménového mena
SOA	6	základné informácie o zóne pre danú doménu
MX	15	poštový server pre danú doménu
RRSIG	46	DNSSEC kľúč
ANY	255	požiadavka na všetky dostupné informácie

Tabuľka 3.2: Typy DNS záznamov (inšpirované z [13])

3.4 DNS protokol

DNS protokol je protokolom aplikačnej vrstvy a slúži na výmenu informácií v celom prostredí DNS. Na transportnej vrstve využíva ako TCP, tak aj UDP protokol, vždy však len jeden pre celý priebeh komunikácie. Primárne využívaným je UDP protokol kvôli menšej réžii. Protokol TCP je potrebný, až keď veľkosť paketu presiahne 512[1] bajtov a ten musí byť rozdelený do viacerých paketov. Túto situáciu dokáže vyriešiť DNS protokol aj pre UDP. Využíva na to rozšírenie EDNS0. Na sieťovej vrstve je potom využívaný IP protokol, ako môžeme vidieť na obrázku 3.1. Komunikácia na úrovni klient-server prebieha pomocou DNS požiadaviek a odpovedí. Tie budú, spolu s dôležitými údajmi z hlavičiek protokolov, popísané ďalej v tejto podkapitole.



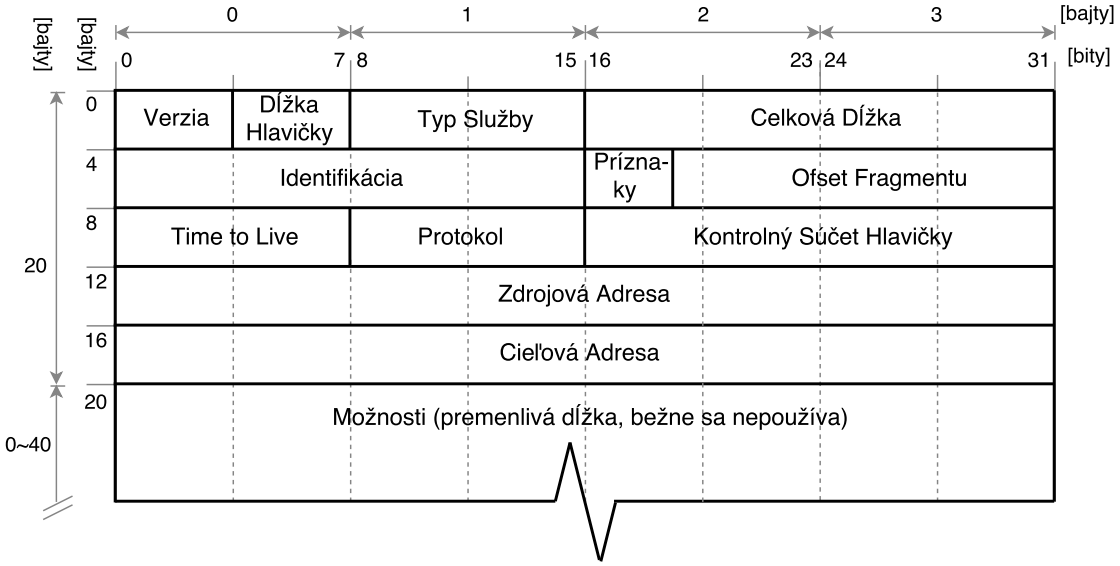
Obr. 3.1: Štruktúra DNS paketu

[1]Všetky IPv4 systémy musia byť schopné prijať paket veľkosti najmenej 576 bajtov. 576 B + 4 B zarovnanie − 60 B maximálna dĺžka IP hlavičky − 8 B dĺžka UDP hlavičky = 512 B.

IPv4 hlavička

IPv4 hlavička (uvedená na obrázku 3.2) má typicky 20 bajtov. Maximálne môže dosiahnuť veľkosti 60 bajtov. To v prípade využitia všetkých 40tich bajtov nastaviteľných IP parametrov nachádzajúcich sa v hlavičke za cieľovou adresou. Položky dôležité z hľadiska testovania slabín systému DNS sú uvedené v nasledujúcom prehľade (viz. dokument RFC 760 [15]):

- Typ služby (ToS) – určuje prednosť paketov.
- Identifikácia – hodnota nastavovaná odosielateľom. Pri fragmentácii majú fragmentované pakety túto hodnotu rovnakú.
- Time to Live – udáva maximálny možný počet „hopov“ paketu. V linuxovom prostredí býva jej hodnota 64, vo windowsovom 128.
- Protokol – určuje, ktorý protokol vyššej vrstvy je v IPv4 zapuzdrený.
- Zdrojová adresa – v prípade, že IPv4 protokol nesie UDP protokol, je možné ju jednoducho podvrhnúť. Pri TCP protokole to možné nie je kvôli tzv. „handshake“.



Obr. 3.2: Hlavička IPv4 protokolu

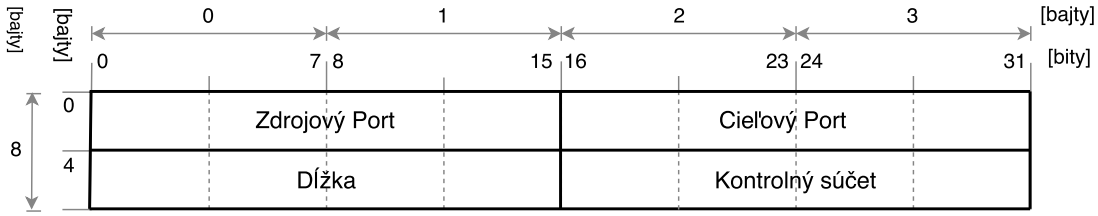
UDP hlavička

UDP (User Datagram Protocol) sa prednostne používa na DNS komunikáciu vďaka jeho nízkej réžii. Nevýhodou je, že nezaručuje úspešné doručenie paketu a odosielateľa neinformuje o doručení. To ale v systéme DNS neprekáža, lebo požiadavka sa dá posielať opakovane, až kým nepríde odpoveď. V minulosti boli DNS pakety malé a nebol problém s fragmentáciou. To už dnes úplne neplatí v prípadoch, keď protokol nesie aj dáta spojené s mechanizmom DNSSEC[2]. UDP hlavička (uvedená na obrázku 3.3) obsahuje iba dva zaujímavé údaje a to (viz. dokument RFC 760 [16]):

[2]DNSSEC (Domain Name System Security Extensions) – zabezpečuje bezpečnú komunikáciu v rámci DNS. Zaisťuje autentizáciu komunikujúcich strán aj integritu dát.

a) Zdrojový port – má význam v prípade, keď označuje port, na ktorom odosielateľ čaká odpoveď.

b) Cieľový port – v prípade DNS serverov to býva port číslo 53.

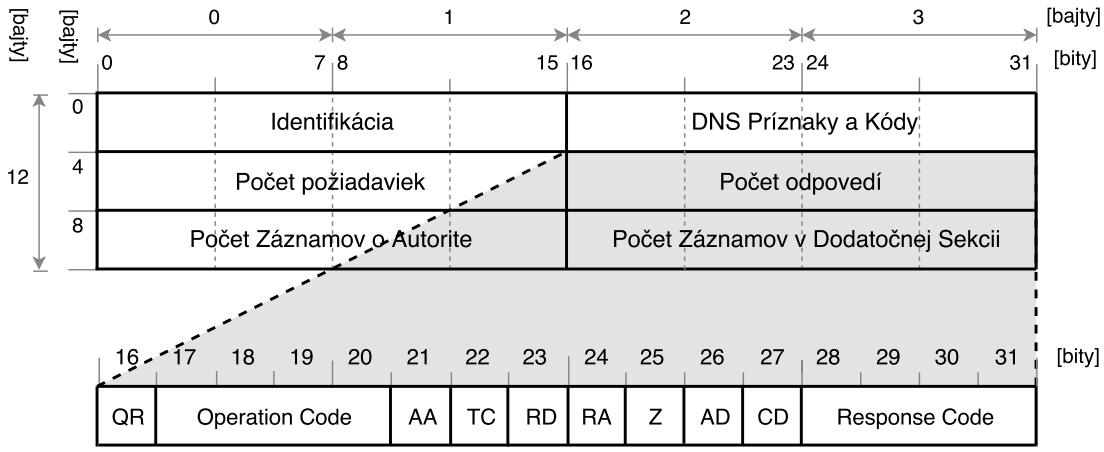


Obr. 3.3: Hlavička UDP protokolu

DNS hlavička

Hlavička DNS protokolu (viz. obrázok 3.4) obsahuje informácie o dátach, ktoré protokol nesie. Okrem svojej identifikácie určuje aj počty požiadaviek a odpovedí, ktoré sú prítomné v dátovej časti protokolu. Druhý a tretí bajt hlavičky je vyhradený pre príznaky a kódy. Z nich sú v tejto práci dôležité tieto (viz. dokument RFC 1035 [13]):

- QR (Query/Response) – určuje či sa jedná o požiadavku („0“), alebo o odpoveď („1“).
- AA (Authoritative Answer) – hodnota „1“ znamená, že odpoveď posiela autoritatívny server pre doménové meno uvedené v sekcii s požiadavkami.
- RD (Recursion Desired) – pre hodnotu „1“ sa menný server snaží DNS požiadavku spracovať rekurzívne.
- RA (Recursion Available) – nastavuje sa v odpovediach. Hodnota „1“ znamená, že menný server dokáže požiadavku spracovať rekurzívne.



Obr. 3.4: Hlavička DNS protokolu

DNS dáta

Dátová časť DNS paketov pozostáva zo štyroch sekcií, ako uvádza publikácia [1]. Sú nimi:

1) Sekcia s požiadavkami (Question Section) – používa sa pri požiadavkách klienta na menný server, ale aj pri odpovediach tohto servera. Menný server do každej odpovede pôvodnú požiadavku skopíruje. Táto sekcia sa skladá ďalej z troch položiek:
   - QNAME – doménové meno zakódované ako je uvedené v tabuľke 3.1.
   - QTYPE, QCLASS – viz. podkapitola 3.3. Niektoré hodnoty sú uvedené v tabuľke 3.2, všetky dostupné hodnoty môžeme nájsť na stránke organizácie IANA [7].
2) Sekcia s odpoveďami (Answer Section) – nachádzajú sa tu odpovede na požiadavku odosielateľa. Formát dát sa prakticky nelíši od záznamov uložených na menných serveroch a je popísaný v podkapitole 3.3.
3) Autoritatívna sekcia (Authority Section) – obsahuje informácie o autoritatívnych serveroch pre požadovanú doménu. Používa sa tu záznam typu NS. Formát dát je rovnaký ako v sekcii odpovedí.
4) Dodatočná sekcia (Additional Section) – obsahuje informácie relevantné v kontexte odpovede na požiadavku.

3.5 Zraniteľnosti DNS

Sieťové útoky využívajúce slabín systému DNS sa dajú rozdeliť na dva typy:

1) Útoky cielené priamo na menné servery – obeťami tohto typu útokov sú menné servery. Typickými predstaviteľmi bývajú útoky typu „DNS spoofing“. Do tejto kategórie patrí napríklad otrávenie cache pamäte menného servera (DNS cache poisoning). Priamo na menné servery môžu byť cielené aj útoky typu odmietnutie služby (Denial of Service, ďalej DoS).
2) Útoky zneužívajúce slabiny DNS – nie sú cielené na žiadny z prvkov DNS infraštruktúry. Tú len zneužívajú vo svoj prospech. Predstaviteľmi tohto typu útokov sú amplifikačný DNS útok a DNS tunelovanie.

Útoky typu DoS

Ich cieľom býva zneprístupnenie niektorej online služby jej zákazníkom. Prebiehajú tak, že útočník posiela veľké množstvo požiadaviek na zariadenie obeti a to nestíha reagovať ani na skutočné požiadavky. Tieto útoky sú jedny z najbežnejších a najľahšie uskutočniteľných, aj preto existujú v mnohých variantách. Často bývajú využívané na zakrytie nejakého zložitejšieho útoku. Na zosilnenie účinku útočníci využívajú tzv. „botnet-y[3]“.

Najväčšie distribuované DoS útoky na koreňové menné servery:

- 6.2. 2007 – okolo 12:00 bol zaznamenaný 2,5 hodinový útok. Po ňom nasledoval ďalší 5 hodinový. Napadnutých bolo 6 koreňových serverov, ale len 2 z nich útok ovplyvnil. Tie v tom čase ešte neimplementovali anycastové adresovanie, ktoré slúži aj ako ochrana pred DDoS útokmi. Údaje o útoku sú zaznamenané v dokumente organizácie ICANN [8].

[3]Botnet – sieť počítačov infikovaných útočníkovým softvérom

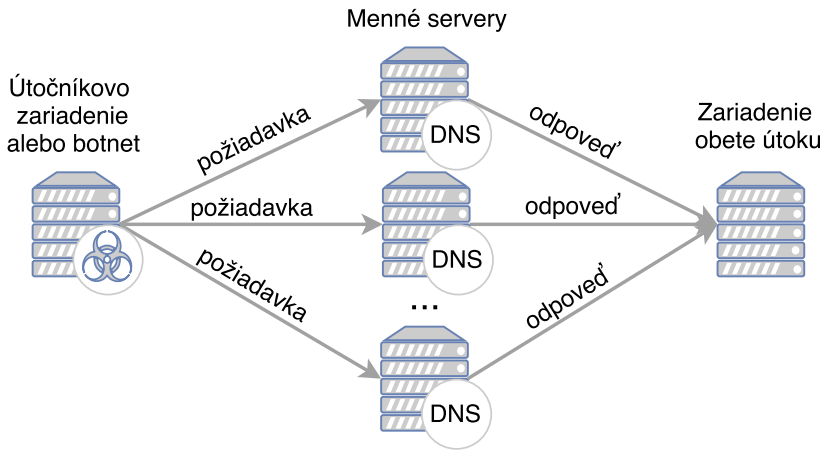
- 30.11. 2015 – z veľkého počtu IPv4 adries boli posielané požiadavky na jedno doménové meno. To spôsobilo timeout-y pre valídnu komunikáciu, ako sa uvádza v zázname o udalosti na koreňových serveroch [17].

Ani tieto veľké útoky nemali výraznejší dopad na komunikáciu na nižších vrstvách DNS.

Variantou tohto útoku, zaujímavou z hľadiska DNS, je amplifikačný útok (DNS amplification attack). Na obrázku 3.5 môžeme vidieť, ako tento útok prebieha. Jeho pointou je asymetria medzi DNS požiadavkami a odpoveďami, keď odpovede majú násobne väčšiu veľkosť. Pomocou napadnutých menných serverov sa dá takto dosiahnuť výrazné zosilnenie (amplifikácia) DoS útoku. Maximálna veľkosť odpovede môže byť až 4096 bajtov s použitím EDNS0 (viz. podkapitola 3.4). Dosiahnuť sa dá dvoma spôsobmi:

- Bez zásahu útočníka – je potrebné pomocou bežných požiadaviek zistiť, ktorá generuje najväčšiu odpoveď.
- So zásahom útočníka – najprv sa podvrhne maximálne dlhý záznam do cache pamäte menného servera a potom sa na tento záznam začnú posielať požiadavky.

Presmerovanie odpovedí z útočníkovho zariadenia na zariadenie obete spôsobuje podvrhnutá IP adresa odosielateľa. To umožňuje transportný protokol UDP, ktorý zdrojové adresy nijako neoveruje.



Obr. 3.5: Schéma amplifikačného útoku na DNS

Otrávenie pamäte cache

Princípom otrávenia pamäte cache je podvrhnutie odpovede, ktorá sa tvári, že prišla z autoritatívneho servera pre požadovanú doménu. Na obrázku 3.6 je popísaný priebeh rôznych variánt útoku. V popise sú fázy útoku očíslované podľa tohto obrázka. Najjednoduchšou variantou útoku je:

1. Útočník pošle požiadavku na menný server. Ten musí podporovať rekurziu (viz podkapitola 3.4). Menný server potom preposiela požiadavku ďalším menným serverom, až pokým sa mu nepodarí zistiť odpoveď. Čas, kým sa mu podarí kontaktovať autoritatívny server pre požadovanú doménu, má k dispozícii útočník. Ten sa zatiaľ snaží vytvoriť odpoveď, akú napadnutý menný server očakáva od autoritatívneho servera. Podmienkami

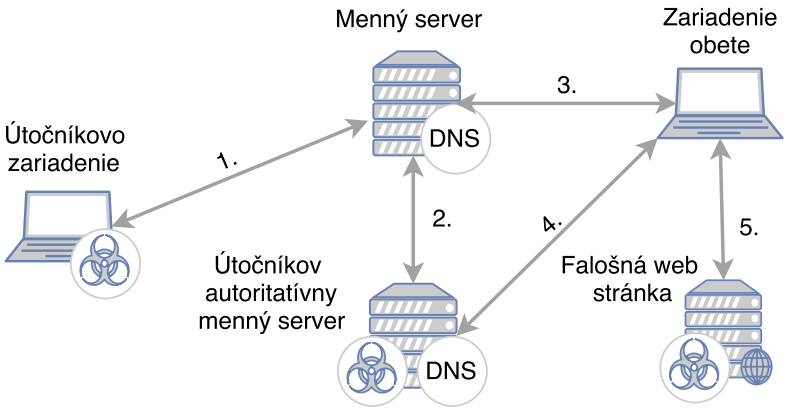
prijatia odpovede z jeho strany, ako uvádzajú vo svojej publikácii [18] Son a Shmatikov, sú:

- Správna zdrojová IP adresa (IP hlavička) – menný server očakáva, že odpoveď príde z adresy autoritatívneho servera, ktorému predtým poslal požiadavku. Túto adresu si vie útočník dopredu zistiť a podvrhnúť. Napr. pomocou programu *nslookup* je možné zistiť názov autoritatívneho servera pre akúkoľvek doménu.
- Správny cieľový port (UDP hlavička) – novšie verzie menných serverov otvárajú komunikáciu s ďalšími servermi na rôznych portoch (nielen na porte 53). Je to bezpečnostné opatrenie proti tomuto typu útoku. Útočník tak musí číslo portu hádať.
- Zhodné DNS identifikačné číslo (DNS hlavička) – ID odpovede sa musí zhodovať s ID požiadavky poslanej menným serverom. Toto číslo musí útočník tiež uhádnuť.

Z uvedeného vyplýva, že útočník musí v odpovedi uhádnuť dve 16 bitové čísla. Aby bolo možné stihnúť vygenerovať dostatočné množstvo falošných odpovedí skôr než príde skutočná, býva tento útok spojený napríklad s DoS útokom na autoritatívny menný server. V prípade, že sa útočníkovi podarí správne uhádnuť parametre odpovede, napadnutý DNS server si ju zapíše do svojej cache pamäte. V odpovedi môže byť napríklad podvrhnutá IPv4 adresa, v prípade požiadavky typu A.

3. Obeť pošle požiadavku na napadnutý menný server. Predpokladajme, že to bude požiadavka typu A na doménové meno, pre ktoré má menný server podvrhnutý záznam. Namiesto skutočnej IPv4 adresy tak obeť dostane falošnú.

5. Falošná IPv4 adresa môže viesť napríklad na útočníkov webový server. Obeť tak môže bez podozrenia zadať citlivé údaje na falošnej webovej stránke.



Obr. 3.6: Schéma otrávenia pamäte cache menného servera

Účinnosť útoku bolo možné zvýšiť s vyšším počtom rovnakých odoslaných požiadaviek. Jedná sa potom o tzv. „narodeninový útok“. Podmienkou jeho uskutočnenia je, aby menný server preposielal ďalej každú požiadavku, ktorú dostane. Dnešné servery už rovnaké požiadavky automaticky nepreposielajú, ale pošlú ďalej len jednu požiadavku.

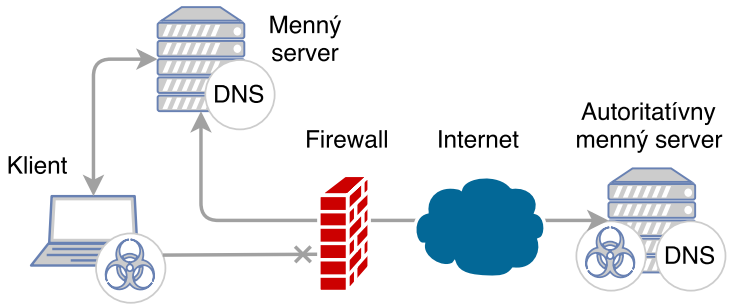
V roku 2008 prišiel Dan Kaminsky s rozsiahlejšou variantou útoku. Mechanizmus je rovnaký, ako bolo popísané vyššie v bode 1. Podstatou je, že falošná informácia sa nenachádza v sekcii odpovedí, ale v dodatočnej sekcii. Útočník pošle napríklad požiadavku

na „neexistujuca-subdomena.domena.com“. V odpovediach potom uvedie v autoritatívnej sekcii, že názov autoritatívneho menného servera pre doménu „domena.com“ je napríklad „utocnikov-menny-server.domena.com“. Nakoniec v dodatočnej sekcii uvedie podvrhnutú IP adresu pre „utocnikov-menny-server.domena.com“. Takto je možné presmerovať, na útočníkom ovládaný menný server, akékoľvek požiadavky smerujúce na doménu (v tomto prípade „domena.com“) a všetky jej subdomény. Komunikácia s falošným menným serverom je znázornená v bodoch 2. a 4. na obrázku 3.6. Informácie o tomto útoku boli čerpané z analýzy Emanuela Petra [14].

Tento typ útoku výrazne obmedzilo rozšírenie mechanizmu DNSSEC. Pokiaľ ho ale neimplementujú všetky články v tzv. „reťazci dôvery“, od klientských programov (resolver) až po autoritatívne menné servery, útok je stále uskutočniteľný.

DNS tunelovanie

Mechanizmy blokujúce sieťovú komunikáciu (napr. firewall) bežne neblokujú DNS protokol. DNS tunelovanie využíva tohto faktu a prostredníctvom DNS paketov prenáša iný protokol. Pozostáva z klientskej aplikácie odosielajúcej dáta a serveru, ktorý je schopný interpretovať tunelovaný protokol.



Obr. 3.7: Schéma DNS tunelovania

Priebeh komunikácie ukazuje obrázok 3.7. Server musí byť autoritatívnym menným serverom pre nejakú kontrolovanú doménu (napr. „kontrolovana-domena.com“). Klientská aplikácia pošle požiadavku na lokálny menný server. V požiadavke je uvedená kontrolovaná doména a na mieste subdomén sú zakódované tunelované dáta (napr. „ADMFW3SEER8.REST04ANKO.kontrolovana-domena.com“). Lokálny server prepošle požiadavku ďalej autoritatívnemu serveru pre kontrolovanú doménu. Ten tunelované dáta interpretuje a v odpovedi môže poslať iné dáta obdobným spôsobom naspäť klientskej aplikácii.

Na kódovanie dát je možné použiť rôzne techniky, v závislosti od varianty tunelovania. Ako príklad môžeme uviesť kódovanie BASE32, najbežnejšie používané v klientských aplikáciách. Princípom je prekódovanie každých 5 bitov vstupných dát na jeden 8-bitový znak zvolenej abecedy. To dáva celkovo abecedu skladajúcu sa z $2^5 = 32$ znakov. V názvoch subdomén je možné použiť všetky alfanumerické znaky a pomlčku, čo je dostatočné množstvo na vytvorenie abecedy.

Kapitola 4

Návrh

Praktická časť vývoja aplikácie zahŕňa 3 vzájomne sa prelínajúce fázy – návrh, implementáciu a testovanie. V tejto kapitole bude ukázaná inšpirácia a vysvetlené dôvody, ktoré viedli k návrhu architektúry aplikácie. V poslednej časti sa bude venovať grafickej stránke používateľského rozhrania.

4.1 Aplikácie podobného charakteru

Aplikácie, ktoré dokážu generovať priamo útoky na DNS alebo iné systémy, bývajú zo služby *Google Play* odstraňované (napr. *dSploit*). Na využitie plnej funkcionality potrebujú spravidla oprávnenia `root`. Na zachytávanie sieťovej komunikácie v systéme Android je vhodný ekvivalent známeho programu *Wireshark*, aplikácia *Shark for Root*. Jej výstupom sú súbory `pcap`, ktoré je možné zobraziť priamo v systéme Android pomocou aplikácie *Shark Reader*. Ďalšími aplikáciami (viz. príloha C) podobného charakteru sú:

- *Magic Tunnel* – aplikácia zameraná na tunelovanie dát cez DNS protokol. Ako jednu z podmienok funkčnosti vyžaduje oprávnenia `root` a nie je dostupná prostredníctvom služby *Google Play*. Mobilná aplikácia slúži na generovanie dát, ktoré sú posielané serveru, ako je popísané v podkapitole 3.5. Slúži na obchádzanie zabezpečenia Wi-Fi sietí. Dosahuje rýchlosti prenosu dát od 1 do 10 KB/s.
- *Packet Generator* – je aplikácia slúžiaca na generovanie TCP, UDP a ICMP paketov pre vzdelávacie a testovacie účely. Základná funkčnosť nevyžaduje `root` oprávnenia.
- *Network Mapper* – slúži ako používateľské rozhranie pre konzolovú aplikáciu *Nmap*. Po spustení aplikácie sa skontrolujú `root` oprávnenia. Pokiaľ sú k dispozícii, stiahne sa hardvérovo zodpovedajúca (podľa tab. 2.2) verzia programu *Nmap*, ktorá sa následne spúšťa pomocou príkazového riadka. Konzolový výstup sa priebežne zobrazuje ako je možné vidieť na obrázku C.1c.

4.2 Architektúra aplikácie

Požiadavky na vymedzenie funkčnosti aplikácie sa dajú zhrnúť do týchto bodov:

- ponuka vybraných útokov, testujúcich známe slabiny systému DNS,
- možnosť zadávania parametrov útokov,

- generovanie paketov na základe zvolených parametrov,
- odosielanie paketov priamo z mobilného zariadenia,
- prehľadné používateľské rozhranie.

Rozhodujúci vplyv na zvolenú architektúru mali dva faktory. Prvým je požiadavka posielať pakety priamo z mobilného zariadenia. Pakety by bolo možné generovať aj pomocou serverovej časti aplikácie. To by však vyžadovalo väčšiu réžiu spojenú s inštaláciou serverovej časti. Ďalšie problémy by spôsobovalo jej umiestnenie v sieti. Na testovanie lokálneho servera v uzavretej sieti by v nej museli byť pripojené obidve časti aplikácie. Tým by sa strácali všetky výhody mobilnej aplikácie a bolo by praktickejšie pracovať pomocou desktopovej nadstavby nad serverom. Druhý faktor vychádza z požiadavky na plnú kontrolu vytvárania paketov. Tá je potrebná napríklad na zadanie falošnej zdrojovej adresy a pod., ako je popísané v podkapitole 3.5. Dá sa dosiahnuť, keď vynecháme zo spracovania paketov jadro operačného systému. Na tento účel slúžia schránky typu `raw`, ktoré má právo vytvárať len proces[1] s CAP_NET_RAW oprávnením.

Voľba implementačného jazyka aplikácie je zhrnutá v nasledujúcom prehľade:

1) Celá aplikácia v jazyku Java – je štandardný spôsob tvorby aplikácií. V tomto prípade som ho však nemohol zvoliť, lebo jazyk Java neposkytuje, kvôli bezpečnosti, potrebné nástroje na vytváranie paketov.

2) Celá aplikácia v C/C++ – táto možnosť existuje, ale je využívaná veľmi zriedka. Na vytvorenie používateľského rozhrania je vhodnejšie použiť na to určený jazyk Java.

3) Časť aplikácie v C/C++ – pre tento prípad najvhodnejšia kombinácia. Dovoľuje napísať jadro aplikácie v C/C++ a na ostatné časti použiť štandardne jazyk Java.

Po zvolení tretej varianty som chcel na komunikáciu s natívnym kódom využiť rozhranie JNI. V rámci aplikácie beží natívny kód spolu s Java kódom v jednom procese. Vzhľadom na vznik procesov popísaný v časti 2.1 nie je možné, bez zásahu do systému, udeliť natívnemu kódu iné práva než má celá aplikácia. Tým pádom sa mu nedajú prideliť oprávnenia na vytváranie `raw` schránok, ktoré Android API neposkytuje. Tieto práva je možné udeliť len novému procesu spustenému pomocou konzoly. Z tohto dôvodu bude nástrojom NDK natívny kód prekladaný do spustiteľných súborov. Tie potom budú vždy s príslušnými právami spúšťané z aplikácie.

Na vytváranie paketov som musel zvoliť jednu z knižníc napísaných v jazyku C. Vyberal som z nasledujúcich variánt:

- `Libpcap`
  - \+ umožňuje vytvárať a zapisovať pakety na sieťové rozhrania,
  - \+ existencia adaptérov poskytujúcich rozhranie tejto knižnice v jazyku Java (napr. `JNetPcap` alebo `Jpcap`),
  - – neposkytuje funkcie na vytváranie hlavičiek a obsahu paketov,
  - – je vhodnejšia na zachytávanie komunikácie než na jej vytváranie,
  - – problematický preklad novších verzií knižnice pre rôzne verzie ABI (viz. tabuľka 2.2).

[1] Pojmy proces a používateľ majú v tomto prípade rovnaký význam. Viz. podkapitola 2.3.

- `Libnet`
  - \+ umožňuje vytvárať pakety až po vrstvu ethernetových rámcov,
  - \+ poskytuje funkcie pre vytváranie hlavičiek všetkých potrebných protokolov (DNS, UDP, IP),
  - − neposkytuje podporu pre vytvorenie dátovej časti DNS protokolu.

Svoju najväčšiu výhodu, možnosť použiť jeden z Java adaptérov, stratila knižnica `Libpcap` nemožnosťou použitia JNI rozhrania. Potrebám tejto aplikácie výrazne viac vyhovuje knižnica `Libnet`.

Na záver stručne zhrniem dôležité body návrhu aplikácie. Kód napísaný v jazyku Java implementuje grafické používateľské rozhranie a komunikuje s jadrom aplikácie napísaným v jazyku C++. Jadro je dostupné prostredníctvom spustiteľných súborov. Na vytváranie paketov je použitá knižnica `Libnet` napísaná v jazyku C.

4.3 Návrh grafického rozhrania

Z grafického hľadiska bolo najťažšie nájsť vhodný spôsob zadávania veľkého množstva parametrov, ktoré sú potrebné pre každý generovaný útok. Táto požiadavka nebýva pre Android aplikácie typická. Každý z programov uvedených v podkapitole 4.1 sa s touto otázkou vyrovnáva iným spôsobom. *Magic Tunnel* používa pre každý vstup vlastný dialóg, ktorý sa zobrazí po kliknutí na pole s názvom parametra (viz. obr. C.1a). *Network Mapper* funguje podobne ako konzolová aplikácia a údaje sa zadávajú pomocou parametrov v jednom textovom poli (viz. obr. C.1c). *Packet Generator* má pre každý parameter vlastné textové pole (viz. obr. C.1b). Ja som zvolil podobný prístup ako *Packet Generator*. Ďalej v tejto podkapitole rozoberiem jednotlivé prvky tvoriace grafické rozhranie aplikácie.

Ovládanie aplikácie

Grafické rozhranie v jednej chvíli zobrazuje vždy len jednu skupinu parametrov reprezentujúcu jeden z podporovaných útokov. Spolu s informáciami o aplikácii tvoria 5 pohľadov, medzi ktorými je možné sa prepínať. Navigáciu medzi nimi umožňujú dva prvky, a to bočný navigačný panel (Navigation Drawer) a tlačidlo (Floating action button) umiestnené vždy v pravom dolnom rohu. Navigačný panel som zvolil za základný prvok navigácie v aplikácii, ktorý zobrazuje zoznam všetkých ponúkaných pohľadov. Tlačidlo umožňuje rýchly prechod na ďalší pohľad a je umiestnené tak, aby nekolidovalo s poliami pre zadávanie textu. Druhou alternatívou bola navigácia pomocou gesta potiahnutia po obrazovke doprava alebo doľava. Gesto potiahnutia doprava by do istej miery kolidovalo s gestom pre zobrazenie bočného panela. Obojsmerné listovanie navyše nie je potrebné vzhľadom na nízky počet pohľadov. Tlačidlo som vybral z dôvodu stability prostredia, v ktorom sa zadávajú parametre útoku. Gestá ostali vyhradené len pre pohyb v rámci jedného pohľadu. Celú navigáciu prehľadne zobrazuje „Mockup“ v prílohe B.

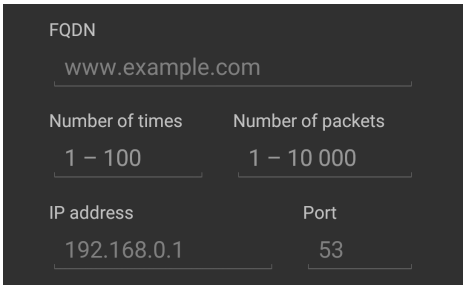
Grafické prvky

Pre tvorbu loga reprezentujúceho aplikáciu som zvolil plochý dizajn (flat design). To znamená, že pri návrhu nie je uvažovaná os „z“ slúžiaca navodeniu efektu priestorovosti. Proces tvorby loga zachytáva obrázok 4.1. Ostatné ikony a obrázky v aplikácii sú prevzaté z voľne

dostupných zdrojov vydaných pod licenciou CC BY 4.0 (napr. Material Design od spoločnosti Google).

Obr. 4.1: Proces vzniku loga aplikácie

Grafické používateľské rozhranie kombinuje tmavé témy z knižníc `Appcompat` a `Holo`. Na zobrazenie zadávania parametrov, v podobe textových vstupov, slúžia textové polia. Sú jediným prvkom využívajúcim štýl knižnice `Holo`. Pokiaľ to je možné, polia vždy vypĺňajú celú dĺžku jedného riadku. Podľa dĺžky vstupu je na riadku buď jedno pole, alebo dve v pomere dĺžok 1:1 alebo 1:2 (viz. obr. 4.2). Každé textové pole obsahuje ukážku správneho vstupu. Polia sú vždy usporiadané do celkov, podľa toho, čo vstupy vyjadrujú (napr. informácie o DNS serveri). V prípade, že význam nie je jasný z označenia nad poľom, stačí



Obr. 4.2: Rozdelenie textových polí na riadku

na označenie kliknúť a zobrazí sa bublina s vysvetlením, prípadne uvedenou implicitnou hodnotou nepovinného poľa. Povinné polia je možné pred vyplnením zobraziť klikom na tlačidlo „START“. Zvýraznia sa červenou farbou s upozornením, že ich je potrebné vyplniť. Všetky prvky rovnakého charakteru využívajú spoločné štýly, v záujme jednotného vzhľadu celej aplikácie. Posledným navrhnutým prvkom bude dialóg. Aplikácia ho využíva v štyroch variantách a to:

- Blokujúci dialóg – je využitý v prípade, že zariadenie používateľa nedisponuje `root` oprávneniami. Tento dialóg o tom informuje a znemožňuje ďalšie používanie aplikácie.
- Informačný dialóg – informuje o nedostupnosti siete alebo o počte odoslaných paketov po úspešnom dokončení príkazu na generovanie útoku.
- Progres dialóg – zobrazuje sa behom odosielania paketov.
- Dialóg na výber súborov – je špeciálny dialóg slúžiaci na prehliadanie súborového systému zariadenia. Graficky rozlišuje adresáre a súbory. Jeho pomocou je možné vybrať akýkoľvek súbor nachádzajúci sa v stromovej štruktúre s koreňovým adresárom `sdcard`.

Kapitola 5

Implementácia

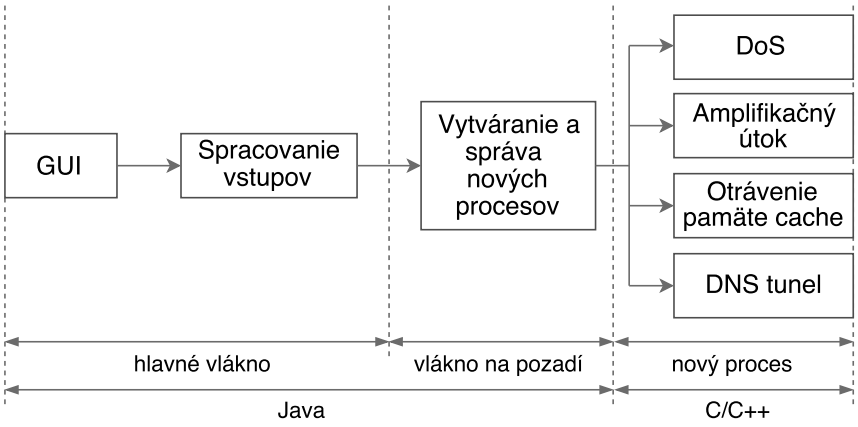
Základná štruktúra aplikácie vyplývajúca z návrhu je naznačená na obrázku 5.1. Celý priebeh implementácie bol rozdelený na 3 časti, a to implementáciu zadávania a spracovania vstupov, implementáciu generovania útokov a komunikáciu medzi nimi. Každú časť bolo možné vyvíjať samostatne. Z hľadiska implementačných jazykov môžeme aplikáciu rozdeliť na 2 časti:

- Implementácia v jazyku Java SE 8 – ako minimálne API som zvolil Android API 16. To zaručuje funkčnosť aplikácie na 95% v súčasnosti používaných zariadení (viz. tabuľka 2.1). Táto časť zahŕňa grafické používateľské rozhranie, spracovanie vstupov a komunikáciu s generátormi útokov. Pôvodne som tu rátal aj s vytváraním dátovej časti DNS protokolu. Vychádzal som zo zistení uvedených v návrhu v časti 4.2 a to, že knižnica `Libnet` neposkytuje podporu na vytváranie týchto dát. Toto riešenie sa však ukázalo byť neefektívne v prípade viacnásobného spúšťania binárnych súborov. Každé spustenie so sebou prináša veľkú réžiu v podobe vytvorenia prístupu na konzolu, spustenia binárneho súboru a prevzatia výstupov. Vytváranie a predávanie všetkých potrebných dát pre jednotlivé útoky naraz je neprehľadné a konceptuálne nezapadá do tejto časti.
- Implementácia v jazyku C++14 – ako cieľovú hardvérovú platformu som zvolil armeabi-v7a. Toto ABI podporuje najväčšia časť súčasných mobilných zariadení. Implementované sú tu generátory jednotlivých útokov. Kód je kompilovaný do podoby spustiteľných konzolových programov.

Na implementáciu boli využité tieto nástroje a prostredia:

- Operačné systémy
  - vývoj aplikácie – Windows 10,
  - testovanie aplikácie – Linux Ubuntu 14.04 LTS (Trusty Tahr), Raspbian Jessie.
- Vývojové prostredie – *Android Studio 1.5.1*
  - automatizovaný preklad – Gradle 2.8,
  - podpora vývoja v natívnom C++ – *Experimental Plugin 0.4.0*.
- Mobilné zariadenie – Xiaomi Mi3
  - ABI – armeabi-v7a,

  - API – 19,
  - verzia Android-u – 4.4.4 KitKat,
  - verzia MIUI – 7 s právami `root`.
- DNS server – *Bind 9*.



Obr. 5.1: Základná štruktúra aplikácie

V priebehu vývoja aplikácie vychádzali pomerne často nové verzie niektorých nástrojov. Prechod medzi nimi nebol vždy priamočiary. Za všetky uvediem verzie experimentálneho plugin-u, ktorý pri prechode na novšiu verziu viackrát zmenil syntax využívaného DSL (viz. podkapitola 2.2).

5.1 Zadávanie a spracovanie vstupov

Vizuálna stránka zadávania vstupov do aplikácie bola popísaná v návrhu v časti 4.3. Každý grafický prvok je popísaný v jednom alebo viacerých XML súboroch. Grafické používateľské rozhranie tvorí 5 pohľadov a bočný navigačný panel. Implementované sú pomocou jednej aktivity a fragmentov. Všetky budú podrobne popísané ďalej v tejto podkapitole.

Základné informácie o aplikácii sú definované v súbore `AndroidManifest.xml`. Okrem iného sú tu uvedené nasledujúce oprávnenia vyžadované časťami aplikácie, kde práva `root` nie sú dostupné:

- `INTERNET`, `ACCESS_NETWORK_STATE`, `ACCESS_WIFI_STATE` – slúžia na zistenie dostupnosti internetového pripojenia a získavanie informácií o sieťových rozhraniach,
- `READ_EXTERNAL_STORAGE` – je potrebné na kontrolu zadaného súboru pri DNS tunelovaní.

Aktivita

Ako už bolo spomenuté, aplikácia implementuje len jednu aktivitu, ktorá sa vytvorí hneď po jej spustení. Pri vytváraní aktivity sa skontroluje dostupnosť `root` oprávnení. V prípade, že nimi zariadenie nedisponuje aplikácia sa zablokuje a zobrazí sa dialóg s touto informáciou. V opačnom prípade sa nainštalujú všetky binárne súbory, nachádzajúce sa v zdrojoch (resources) aplikácie, do adresára `/data/data/com.michalbeder.dnsattacks/files/`. Za

behu aktivita spravuje navigáciu medzi fragmentami prostredníctvom svojej zanorenej triedy `Fragments`. V nej sa koordinuje ovládanie pomocou navigačného panela a tlačidla. V grafickom zobrazení aktivity sa vždy na rovnaké miesto vloží príslušný fragment pomocou komponenty `SupportFragmentManager`.

Fragmenty

Zatiaľ čo sa aktivita stará o navigáciu, vo fragmentoch prebieha komunikácia s používateľom a spracovanie vstupov. Každý z piatich pohľadov je implementovaný pomocou samostatného fragmentu. Všetky fragmenty sú odvodené od triedy `BaseFragment`, ktorá obsahuje túto spoločnú funkcionalitu:

- Kontrola internetového pripojenia – prebieha vždy pred vytvorením vlákna na pozadí a spustením binárnych súborov. V prípade, že pripojenie nie je dostupné zobrazí sa oznamovací dialóg a program vykonávajúci príslušný útok sa nespustí. Pokiaľ pripojenie vypadne počas prebiehajúceho útoku, aplikácia oznámi počet neúspešne odoslaných paketov.
- Zobrazovanie nápovedy – po kliknutí na niektoré textové polia sa zobrazí nápoveda s podrobnejšími informáciami ku konkrétnym vstupom.
- Skrytie softvérovej klávesnice – je potrebné pri každom spustení útoku (kliknutím na tlačidlo START).
- Kontrola vstupov – prebieha na rovnakom mieste ako kontrola internetového pripojenia. Konkrétne sa kontroluje syntax zadaného doménového mena a správnosť formátu IP adresy. V prípade neplatných vstupov je na to používateľ upozornený.

Pri vzniku inštancie fragmentu sa všetky potrebné prvky grafického rozhrania pripoja k jeho inštančným premenným pomocou knižnice `Butter Knife`. Takto je možné priamo získavať vstupy, zobrazovať nápovedy atď. a vyhnúť sa často opakovaným úsekom kódu.

Druhá dôležitá úloha fragmentov je spracovanie získaných vstupov. Pre každý fragment je potrebné na základe vstupov sformulovať konzolový príkaz v tomto tvare:

```
<príkaz> → cesta/<program> <zoznam parametrov>
<zoznam parametrov> → <parameter> hodnota <zoznam parametrov>
<zoznam parametrov> → ϵ
<program> = {dos, ampli, spo, tun}
<parameter> = {-d, -i, -t ...}
cesta = /data/data/com.michalbeder.dnsattacks/files
```

Prvým parametrom v príkaze je vždy parameter „-i“ nasledovaný IP adresou sieťového rozhrania, z ktorého sa majú odosielať generované pakety. Príkazy zostavujú triedy implementujúce rozhranie `INativeCommand`. V týchto triedach sú uložené aj implicitné hodnoty vstupov.

5.2 Vytváranie a správa nových procesov

Programy s útokmi sa spúšťajú pomocou knižnice `Root Tools`. Po vytvorení konzolového príkazu, v podobe textového reťazca, sa v hlavnom vlákne zobrazí dialóg progresu a na

pozadí sa spustí nové vlákno. Otvorí sa nová konzola, na ktorú sa ako prvý pošle nasledujúci príkaz:

```
export LD_LIBRARY_PATH=/data/data/com.michalbeder.dnsattacks/
files:$LD_LIBRARY_PATH
```

Po jeho vykonaní môže následne spustený program využívať knižnicu `Libnet.so` nachádzajúcu sa v uvedenej lokácii. Po úspešnom ukončení programu sa jeho konzolový výstup vráti do hlavného vlákna aplikácie a zobrazí sa používateľovi v novom dialógu.

Prebiehajúci útok je možné zastaviť v dialógu progresu tlačidlom CANCEL. V tom prípade sa zastaví prebiehajúci príkaz na pozadí a zavrie sa konzola. Spustený program sa ukončí volaním príkazu *kill* spolu s názvom programu. Používateľ je o úspešnosti prerušenia útoku informovaný pomocou tzv. „Toast“ správy.

5.3 Generovanie útokov

Kvôli efektivite a konzistentnosti je celá funkcionalita týkajúca sa generovania útokov implementovaná v jazyku C++. Využívaná je pri tom knižnica `Libnet` napísaná v jazyku C. V rámci prehľadnosti je každý útok kompilovaný do samostatného binárneho súboru. Vstupom každého takéhoto programu sú používateľove vstupy predané prostredníctvom parametrov z Java kódu, tak ako bolo popísané v minulej podkapitole. Na základe nich sa vytvoria a posielajú príslušné pakety.

Plugin *Experimental Gradle* vo svojej súčasnej fáze vývoja nepostačoval potrebám prekladu, preto bol využívaný len pre podporu vývoja (syntax, „IntelliSense“ atď.). Preklad pomocou nástroja NDK umožňujú súbory `Android.mk`. Podrobnejší popis sa nachádza v časti 2.2. Adresárová štruktúra musí dodržiavať nasledujúcu schému:

```
main ..... koreňový adresár pre zdrojové súbory aplikácie
├── jni ..... koreňový adresár pre všetky zdrojové súbory natívneho kódu
├── libs ..... cieľový adresár prekladu nástroja NDK
│   ├── armeabi-v7a ..... obsahuje preložené súbory pre platformu armeabi-v7a
│   └── ... ..... adresáre ďalších hardvérových platforiem
├── res ..... zdroje aplikácie
│   ├── raw ..... musí obsahovať všetky binárne súbory používané v aplikácii
│   └── ... ..... ostatné adresáre zdrojov
└── ... ..... ostatné adresáre aplikácie
```

V súbore `Application.mk`, nachádzajúcom sa v adresári `jni`, sa definuje, pre ktorú cieľovú hardvérovú platformu sa majú zdrojové kódy kompilovať. Výsledky prekladu pomocou NDK sa uložia do adresára `/main/libs/armeabi-v7a` a je ich potrebné manuálne prekopírovať do adresára `raw`. Odtiaľ ich potom aplikácia dokáže nainštalovať na cieľové zariadenie. Tieto úkony automatizujú jednoduché skripty `build_native.bat` a `copy.bat` napísané pre platformu Windows.

Generovanie paketov

Rozhranie potrebné na zostavenie paketu poskytuje trieda `DNSPacketBuilder`. Funguje ako adaptér pre knižnicu `Libnet` a dopĺňa chýbajúcu funkcionalitu vytvárania dátovej

časti DNS protokolu. Parametre hlavičiek protokolov reprezentujú štruktúry `DnsHeader_t`, `UdpHeader_t` a `IpHeader_t` (podrobnejšie informácie o hlavičkách sa nachádzajú v časti 3.4). Niektoré hodnoty sú nastaviteľné, iné poskytuje automaticky trieda `DNSPacketBuilder`:

- DNS hlavička
  - Identifikácia – nastavuje používateľ, implicitne je nastavená hodnota 0x0000.
  - Príznaky – pre požiadavku sa nastaví iba bit RD. Tomu zodpovedá hexadecimálna hodnota 0x0100. Pre odpoveď sú nastavené bity QR, AA, RD a RA zodpovedajúce hodnote 0x8580.
  - Počty záznamov v jednotlivých sekciách – automaticky počítané z vložených údajov.
- UDP hlavička
  - Zdrojový port, cieľový port – nastaviteľné hodnoty. Implicitne nastavené na 1024 resp. 53.
- IP hlavička
  - Typ služby – nastavený na hodnotu 0x00. Táto hodnota je štandardne používaná DNS protokolom a znamená najväčšie úsilie doručenia (best effort).
  - Identifikácia – nastaviteľná hodnota. Implicitne 0x0000.
  - TTL – nastavený na hodnotu 128.
  - Protokol – nastavený na hodnotu 17, ktorá označuje UDP protokol.
  - Zdrojová IP adresa, cieľová IP adresa – nastavuje používateľ.

Pred odoslaním paketu na sieťové rozhranie je potrebné zostaviť hlavičky v presnom poradí, od najvyššej vrstvy po najnižšiu. To zabezpečuje funkcia `buildPacket()`. Túto funkciu je potrebné volať aj po ľubovolných zmenách dát, ktoré chceme, aby sa prejavili v odosielaných paketoch. Do ethernetovej hlavičky nie je potrebné nijako zasahovať, doplní ju knižnica `Libnet`.

Rozhranie umožňujúce tvorbu dátovej časti DNS protokolu poskytuje trieda `DNSData`. Používateľovi dovoľuje pridávať jednu alebo viac požiadaviek resp. odpovedí. Tie sú definované pomocou týchto štruktúr (význam položiek v štruktúrach je vysvetlený v časti 3.3):

- `Query_t` – QNAME, QTYPE.
- `Answer_t` – NAME, TYPE, TTL, RDATA.

Všetky uvedené hodnoty nastavuje používateľ. QCLASS resp. CLASS je automaticky nastavovaná na hodnotu 0x0001 (trieda internet). Hodnoty QNAME a NAME sú spracované spôsobom, aký ukazuje tabuľka 3.1. V odpovediach sa môže v tomto poli namiesto celého doménového mena použiť ukazovateľ na toto meno uvedené v zázname s požiadavkou. Ukazovateľ má k dispozícii 16 bitov, z ktorých prvé 2 musia byť nastavené na číslo „1“. Ostatných 14 bitov predstavuje počet bajtov od začiatku DNS hlavičky až po miesto, na ktorom je doménové meno uložené.

Vstupy si každý z programov spracuje individuálne. Následne vytvorí inštanciu triedy `Attacks`, ktorej predá spracované parametre. Táto trieda zapuzdruje všetky dostupné útoky. Pakety vytvára pomocou rozhraní tried a štruktúr popísaných vyššie v tejto časti. Jednotné formátovanie výstupov zabezpečuje pomocou funkcie `printOutput()`.

DoS útok

Prvým z algoritmov na testovanie zraniteľností DNS je generátor DoS útoku. Má minimum nastaviteľných parametrov a slúži na rýchle a jednoduché testovanie. Nastaviť sa dá IP adresa DNS servera, na ktorý majú generované pakety smerovať. Voliteľné sú nastavenia počtu paketov a počet iterácií v ktorých sa pakety posielajú. Medzi každou iteráciou je nastavený časový interval 100 mikrosekúnd. Aby tento útok mohol spôsobiť reálne problémy serveru DNS musel by súčasne prebiehať na viacerých zariadeniach. V prípade potreby je možné pakety posielať opakovane, najviac však 1 000 000 na jedno spustenie.

Pakety sú nastavené ako požiadavky typu A na doménové meno „www.google.com“. IP a DNS identifikácie sa pre každý paket menia. Zdrojová adresa nie je podvrhnutá a cieľový port je automaticky nastavený na štandardný DNS port číslo 53. Ukážka takto vygenerovaného paketu je uvedená na obrázku 5.2.

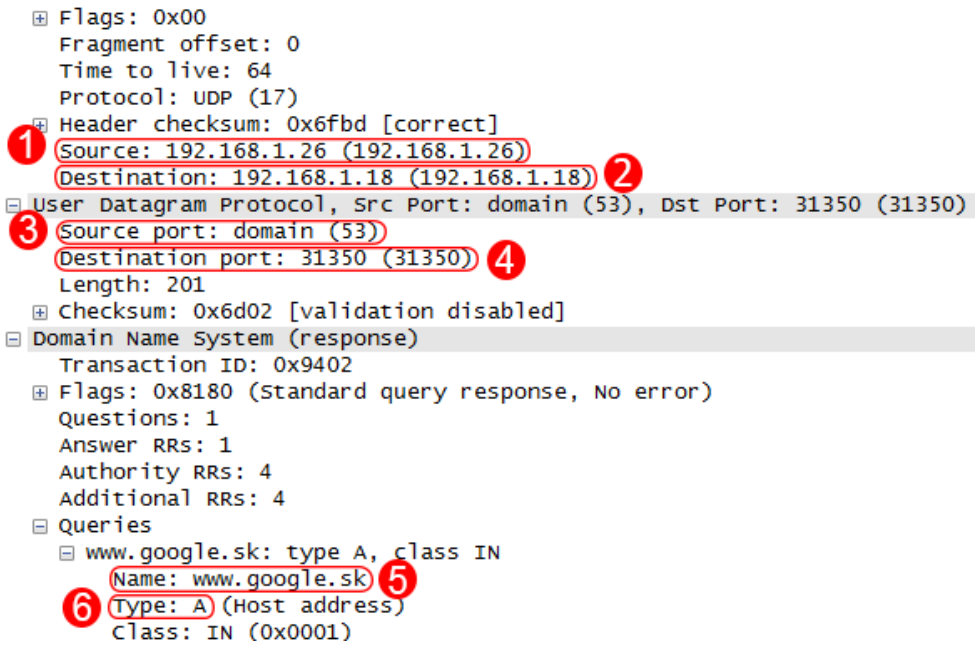
```
⊟ Internet Protocol Version 4, Src: 192.168.1.11 (192.168.1.11), Dst: 192.168.1.17 (192.168.1.17)
    Version: 4
    Header length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT
    Total Length: 60
    Identification: 0xf64d (63053)
  ⊞ Flags: 0x00
    Fragment offset: 0
    Time to live: 128
    Protocol: UDP (17)
  ⊞ Header checksum: 0xc0f6 [correct]
    Source: 192.168.1.11 (192.168.1.11)
    Destination: 192.168.1.17 (192.168.1.17)
⊟ User Datagram Protocol, Src Port: 7732 (7732), Dst Port: domain (53)
    Source port: 7732 (7732)
    Destination port: domain (53)
    Length: 40
  ⊞ Checksum: 0xd9bb [validation disabled]
⊟ Domain Name System (query)
    Transaction ID: 0xf564
  ⊞ Flags: 0x0100 (Standard query)
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ⊟ Queries
    ⊟ www.google.com: type A, class IN
        Name: www.google.com
        Type: A (Host address)
        Class: IN (0x0001)
```

Obr. 5.2: Ukážka požiadavky pri DoS útoku (z programu *Wireshark*)

Amplifikačný útok

Základom amplifikačného útoku je podvrhnutie zdrojovej IP adresy v požiadavke posielanej útočníkom. Namiesto skutočnej adresy sa použije IP adresa zariadenia obete. Takýmto spôsobom sa dá určiť aj port, na ktorý budú DNS odpovede prichádzať. V ostatných atribútoch sa požiadavky nijako nelíšia od bežných legitímnych DNS požiadaviek. Zneužitý DNS server tak neposiela odpovede naspäť útočníkovi, ale obeti. Na obrázku 5.3 je uvedená časť takejto odpovede zachytená na zariadení obete útoku. Čísla označujú údaje, ktoré boli zadané používateľom (útočníkom) v aplikácii:

1. Zdrojová adresa – v tomto prípade to je adresa DNS servera, z ktorého bola odpoveď poslaná.
2. Cieľová adresa – adresa zariadenia obete. V aplikácii sa zadáva ako „Target cpu“.
3. Zdrojový port – štandardne je to port 53 používaný systémom DNS. Pre špeciálne prípady je možné nastaviť aj iné číslo portu.
4. Cieľový port – je port zariadenia obete, na ktorý smerujú odpovede DNS servera.
5. Doménové meno – môže sa jednať o plne kvalifikované doménové meno alebo doménu v závislosti od typu požadovaného záznamu.
6. Typ požiadavky – v aplikácii sa dajú okrem typu A nastaviť aj typy CNAME, SOA, MX, RRSIG a ANY.



Obr. 5.3: Ukážka odpovede pri amplifikačnom útoku (z programu *Wireshark*)

Otrávenie pamäte cache

Otrávenie pamäte cache menného servera prebieha presne tak, ako bolo popísané v časti 3.5. Algoritmus generovania útoku prebieha v nasledujúcich krokoch:

- Vytvorenie požiadavky – zo vstupných dát sa použije adresa a port cieľového DNS servera a k nim sa automaticky pridá typ požiadavky A. Z týchto údajov sa zostaví paket s požiadavkou.
- Vytvorenie odpovede – najprv sa musí pridať pôvodná požiadavka do sekcie požiadaviek. Ďalšie tri sekcie (viz. podkapitola 3.4) sa pridávajú v závislosti na variante útoku.

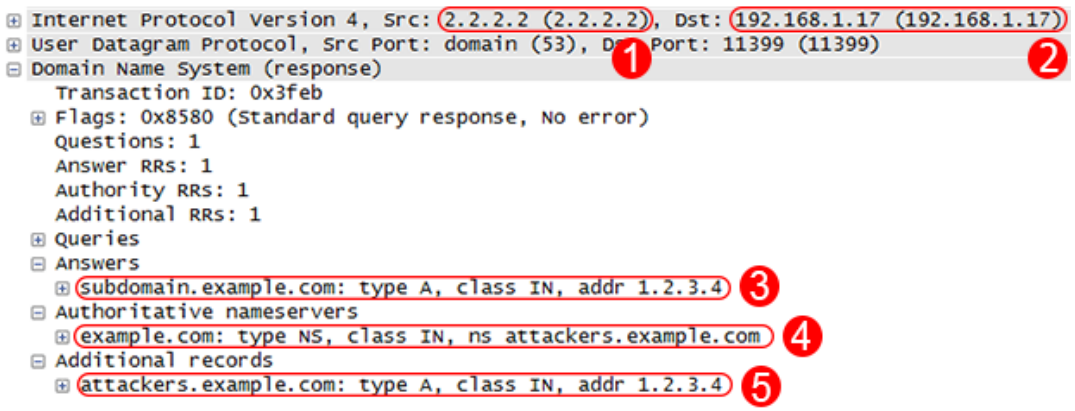
- Odosielanie vytvorených požiadaviek a odpovedí – ich počet volí používateľ. V prípade voľby väčšieho množstva požiadaviek sa jedná o narodeninový útok. Behom odosielania odpovedí sa menia DNS identifikácia a číslo zdrojového portu.

Implementácia podporuje dve varianty útoku:

- Základnú variantu s podporou podvrhnutia IP adresy pre konkrétne doménové meno. Do výsledného paketu sa pridá iba sekcia s odpoveďami. Na obrázku 5.4 to zodpovedá záznamu číslo 3, čísla 4 a 5 by sa v odpovedi nenachádzali. V prípade úspechu útoku by boli presmerované všetky požiadavky na doménové meno „subdomain.example.com“ na adresu „1.2.3.4“.

- Kaminského variantu útoku, ktorá nastane ak používateľ vyplní aj voliteľné parametre v aplikácii (Optional spoofed data). Príklad odpovede generovanej touto variantou je uvedený na obrázku 5.4. Táto odpoveď bola zachytená na zariadení, na ktorom fungoval napadnutý DNS server. Vyznačené údaje v odpovedi majú tento význam:

  1. Zdrojová adresa – jedná sa o adresu autoritatívneho servera pre doménové meno uvedené v bode 3. („subdomain.example.com“). Adresu zadáva používateľ.
  2. Cieľová adresa – adresa napadnutého menného servera.
  3. Sekcia s odpoveďami – obsahuje doménové meno prevzaté z požiadavky a podvrhnutú adresu zadanú používateľom.
  4. Autoritatívna sekcia – napadnutému mennému serveru hovorí, že autoritatívny server pre doménu „example.com“ je útočníkov server „attackers.example.com“.
  5. Dodatočná sekcia – upresňuje adresu útočníkovho menného servera. Pokiaľ si ju napadnutý server uloží do svojej cache pamäte, budú všetky požiadavky na doménu „example.com“ a všetky jej subdomény presmerované na útočníkov menný server na IP adrese „1.2.3.4“.



Obr. 5.4: Ukážka generovaných odpovedí pri otrávení pamäte cache (z programu *Wireshark*)

DNS tunel

Aplikácia slúži ako klientská časť systému DNS tunelovania. Dokáže tak generovať dáta napríklad pre detekčné algoritmy. Základom generovaných dát je používateľom v aplikácii

vybraný súbor. Konzolovému programu je predaná jeho absolútna cesta. Práca so súborom prebieha v triede `FileReader`. Vzhľadom na obmedzenú operačnú pamäť mobilných zariadení sa súbor načítava po častiach o veľkosti 1 kB. Načítané binárne dáta sú hneď prekódované pomocou kódovania BASE32. Rozhranie na kódovanie binárnych dát poskytuje trieda `Base32`. Používa pri tom túto abecedu:

ABCDEFGHIJKLMNOPQRSTUVWXYZ23456789

Na zarovnanie bajtov sa namiesto typicky používaného znaku „=“ musí použiť znak „0“, ktorý je povolený ako koncový znak subdomény. Okrem súboru používateľ zadáva ešte tieto údaje:

- Doménové meno – malo by byť také, pre ktoré má používateľ pod kontrolou príslušný autoritatívny menný server. Pre testovacie účely je ale možné použiť akékoľvek.
- Oneskorenie – aby sa zabránilo zahodeniu tunelovaných paketov na sieťových zariadeniach dá sa zadať hodnota oneskorenia medzi paketmi.
- IP adresa a port DNS servera – cieľová stanica pre generované pakety. Bežne je cieľom DNS server v lokálnej sieti, ale nemusí byť. Cieľová stanica aj port sa dajú nastaviť podľa potreby.

Na zadaný DNS server sa potom odosielajú takto zostavené požiadavky:

a) Typ – používajú sa výhradne požiadavky typu A.

b) Meno (QNAME) – z prekódovaných binárnych dát sa vytvoria subdomény náhodnej dĺžky. Tie sa v náhodnom počte pripoja pred zadané doménové meno. Celý proces sa riadi pravidlami pre tvorbu doménových mien, ktoré sú uvedené v časti 3.2.

c) DNS a IP identifikácia – menia sa s každou novovytvorenou požiadavkou.

Požiadavky sa posielajú dovtedy, kým nie je odoslaný celý súbor. Aby bolo možné jednoznačne určiť koniec súboru, ako posledná sa vždy pošle požiadavka obsahujúca doménové meno v tvare „A1.<zadané doménové meno>“. Jedinečnosť subdomény „A1“ je zaručená tým, že znak „1“ sa nenachádza v abecede kódovania BASE32.

5.4 Testovanie

Najväčšie ťažkosti pri testovaní spôsobovala, tak ako aj pri vývoji, nevyhnutnosť `root` oprávnení. Tie nie sú v mobilných zariadeniach bežne prístupné a vyžadujú si ich úpravu, ktorá narúša bezpečnostný model Android-u. Aj preto nie je ľahko dostupný väčší počet takýchto zariadení, na ktorých by bolo možné aplikáciu testovať. Túto nevýhodu som sa snažil eliminovať oddeleným testovaním častí aplikácie. Časť napísaná v jazyku Java nevyžadovala `root` oprávnenia a bola ľahko testovateľná na mobilných zariadeniach s rôznymi verziami systému Android a rôznymi rozmermi displeja. Konzolové programy napísané v jazyku C++ boli zase testovateľné samostatne aj na iných platformách. Celú aplikáciu so všetkými jej komponentami som testoval prevažne na zariadení Xiaomi Mi3 s dostupnými právami superpoužívateľa.

Pokiaľ nebola hotová celá potrebná infraštruktúra v časti aplikácie napísanej v jazyku Java, prebiehalo testovanie C++ programov pomocou konzolového emulátoru v Android-e.

Práca s ním nie je príliš praktická, preto som používal skripty, ktoré ju čiastočne automatizovali. Paralelne s tým som časti C++ kódu vyvíjal a testoval na systémoch Windows 10 a Linux Ubuntu 14.04. Pravidelne som musel tento kód prekladať pomocou nástroja NDK a nasadzovať ho na systém Android. Dôvodom je neprítomnosť niektorých knižníc a odlišnosť mobilných a desktopových hardvérových platforiem. Naopak ale kód fungujúci na mobilnej platforme bol spravidla preložiteľný a funkčný aj na desktopových platformách.

Spúšťanie programov z konzoly v systéme Android a ich vynútené ukončenie som testoval pomocou jednoduchého „mock“ programu skompilovaného pre všetky mobilné hardvérové platformy uvedené v tabuľke 2.2. Ten sa pokúsil vytvoriť `raw` schránku a v prípade úspechu vygeneroval výpis na štandardný výstup. Pre prípad testovania predčasného ukončenia programu existovala varianta tohto programu s nekonečnou slučkou a pravidelným uspávaním na jeho konci. Takto bolo možné odladiť skoro celú časť aplikácie napísanú v jazyku Java.

Na overenie funkčnosti jednotlivých útokov som používal podobnú topológiu, akú znázorňujú obrázky v podkapitole 3.5. Pozostávala z troch komponent:

1. Zariadenia útočníka – mobilné zariadenie s nainštalovanou aplikáciou.

2. DNS servera – použitý bol server *Bind9* s povolenou rekurziou a vypnutou autentizáciou pomocou DNSSEC.

3. Zariadenia obete – zariadenie zachytávajúce prichádzajúce pakety pre budúcu analýzu.

V úlohe DNS servera a obete sa striedali zariadenia s operačnými systémami Windows 10, Linux Ubuntu 14.04 LTS (Trusty Tahr) a Raspbian Jessie. Na každom zariadení bol pri testovaní spustený program na zachytávanie sieťovej komunikácie. Jednalo sa vždy o jeden z programov *Wireshark*, *Shark for Root* a *tcpdump* v závislosti od operačného systému. Na analýzu zachytených dát som spravidla používal program *Wireshark*.

5.5 Možnosti ďalšieho vývoja

Budúce rozširovanie aplikácie by sa mohlo uberať dvoma smermi. Prvým z nich je pridanie nových typov útokov. Do časti aplikácie napísanej v jazyku Java je možné bez problémov pridať akýkoľvek typ útoku. Pokiaľ to nebude potrebné, nemusí byť ani implementovaný v inom jazyku než Java. Čo sa týka časti napísanej v C++ je možné jednoducho použiť existujúce rozhrania a implementovať ďalšie programy testujúce zraniteľnosti DNS. Implementáciu súčasných programov by vylepšila možnosť využiť TCP protokol alebo podpora posielania DNS paketov až do veľkosti 4096 B pomocou EDNS0. Takto by amplifikačný útok bol automaticky schopný využívať maximálny koeficient amplifikácie, čo pre súčasné účely nebolo potrebné. Pre testovanie iných systémov než DNS je potrebné pridať podporu potrebného protokolu.

Druhý smer, ktorým by sa aplikácia mohla rozšíriť je pridanie možnosti zachytávania a analýzy DNS komunikácie. Takto by bolo možné napríklad automaticky zisťovať autoritatívne servery pre rôzne doménové mená a v aplikácii graficky zobrazovať náhľady posielaných a prijímaných paketov. Vďaka spôsobu implementácie je možné do aplikácie jednoducho zakomponovať aj rôzne programy a knižnice, ktoré boli vyvinuté v jazykoch C/C++ pre iné platformy. Príkladmi vhodných programov sú už v práci spomínaný *tcpdump* alebo *Nmap*.

Kapitola 6

Záver

V rámci tejto práce bol ukázaný spôsob, akým je možné implementovať Android aplikáciu, ktorá má plne pod kontrolou tvorbu paketov nesúcich DNS protokol. Úlohu komplikujú bezpečnostné modely jazyka Java a systému Android. Ten štandardne neposkytuje podporu pre manuálne vytváranie paketov. Preto som zvolil pre Android aplikácie netypický návrh, ktorý zahŕňa dva implementačné jazyky. Výhodou tohto prístupu bola možnosť použitia overených nástrojov implementovaných v jazyku C. Z knižníc `Libpcap` a `Libnet` bola pre účely aplikácie vhodnejšia druhá menovaná. `Libpcap` by naopak mohla tvoriť základ pre ďalší vývoj. Na zlepšení podpory vývojových prostredí pre kompiláciu už existujúcich knižníc a implementáciu v jazyku C++ sa stále pracuje. Podľa dostupných informácií by mal byť takýto vývoj v budúcnosti výrazne priamočiarejší a jednoduchší.

Výsledná aplikácia nazvaná *DNS Attacks* je schopná generovať vybrané útoky na systém DNS. Pri každom útoku sú zvolené kľúčové parametre, ktoré môže používateľ nastavovať. Tie sú prehľadne zoradené podľa významu a zobrazené prostredníctvom grafického používateľského rozhrania. Význam niektorých vstupov upresňujú nápovedy, ktoré sa zobrazia po kliknutí na textové pole umiestnené nad príslušným vstupom. Medzi jednotlivými útokmi je navrhnutá jednoduchá a rýchla navigácia.

Vzhľadom na nedostupnosť `root` oprávnení v systéme Android nie je možné aplikáciu používať na akomkoľvek zariadení. Používateľ si musí tieto oprávnenia zabezpečiť pre svoje konkrétne zariadenie. Vnútorná reprezentácia ostáva, na rozdiel napríklad od aplikácie *Network Mapper*, používateľovi skrytá. Všetky potrebné binárne súbory sa nainštalujú pri prvom spustení aplikácie na pozadí a používateľ sa s nimi nedostane priamo do kontaktu. Pomocou aplikácie je možné otestovať konkrétny DNS server alebo vygenerovať dáta pre detekčné algoritmy. V neposlednom rade môže tiež aplikácia slúžiť na edukatívne účely. Tými môžu byť na jednej strane pochopenie zraniteľností DNS a na druhej cesta akou je možné podobné aplikácie implementovať.

Literatúra

[1] Aitchison, R. *Pro DNS and BIND 10*. Books for professionals by professionals, Apress, 2011. ISBN 9781430230489.

[2] *Android Developers* [online]. Google, Inc.; Open Handset Alliance, 2016 [cit. 2016-04-10]. Dostupné z: <http://developer.android.com/about/dashboards/>.

[3] *Android Interfaces and Architecture* [online]. Google, Inc.; Open Handset Alliance, 2016 [cit. 2016-04-15]. Dostupné z: <https://source.android.com/devices/>.

[4] *Android NDK* [online]. Google, Inc.; Open Handset Alliance, 2016 [cit. 2016-04-09]. Dostupné z: <http://developer.android.com/ndk/>.

[5] *Experimental Plugin User Guide* [online]. Google, Inc.; Open Handset Alliance, 2016 [cit. 2016-04-12]. Dostupné z: <http://tools.android.com/tech-docs/new-build-system/gradle-experimental/>.

[6] *System and kernel security* [online]. Google, Inc.; Open Handset Alliance, 2016 [cit. 2016-04-12]. Dostupné z: <https://source.android.com/security/>.

[7] *Domain Name System (DNS) Parameters* [online]. IANA, 2016 [cit. 2016-04-19]. Dostupné z: <http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml/>.

[8] Factsheet Root server attack on 6 February. 2007 [cit. 2016-05-15]. Dostupné z: <https://www.icann.org/en/system/files/files/factsheet-dns-attack-08mar07-en.pdf>

[9] Kabelová, A.; Dostálek, L. *Velký průvodce protokoly TCP/IP a systémem DNS*. Computer Press, 2008. ISBN 978-80-251-2236-5.

[10] Liu, F. *Android Native Development Kit Cookbook*. Quick answers to common problems, Packt Publishing, 2013. ISBN 9781849691505.

[11] Matoušek, P. *Síťové aplikace a jejich architektura*. VUTIUM, 2014. ISBN 978-80-214-3766-1.

[12] Mednieks, Z.; Dornin, L.; Meike, G.; aj. *Programming Android: Java Programming for the New Generation of Mobile Devices*. O'Reilly Media, 2012. ISBN 9781449358471.

[13] Mockapetris, P. : Domain names - implementation and specification. STD 13, RFC Editor, November 1987, 10.17487/RFC1035. Dostupné z: <http://www.rfc-editor.org/rfc/rfc1035.txt>

[14] Petr, E. : An analysis of the DNS cache poisoning attack. 2009 [cit. 2016-05-15]. Dostupné z: <https://www.nic.cz/files/labs/DNS-cache-poisoning-attack-analysis.pdf>

[15] Postel, J. : DoD standard Internet Protocol. RFC 760, RFC Editor, January 1980, 10.17487/RFC0760). Dostupné z: <http://www.rfc-editor.org/rfc/rfc760.txt>

[16] Postel, J. : User Datagram Protocol. STD 6, RFC Editor, August 1980, 10.17487/RFC0768. Dostupné z: <http://www.rfc-editor.org/rfc/rfc768.txt>

[17] *Events of 2015-11-30* [online]. Root Server Operators, 2015 [cit. 2016-04-05]. Dostupné z: <http://www.root-servers.org/news/events-of-20151130.txt>.

[18] Son, S.; Shmatikov, V. *Security and Privacy in Communication Networks: 6th Iternational ICST Conference, SecureComm 2010, Singapore, September 7-9, 2010. Proceedings*. Jajodia, Sushil and Zhou, Jianying. Springer Berlin Heidelberg, 2010. The Hitchhiker's Guide to DNS Cache Poisoning, s. 466–483. Dostupné z: <http://dx.doi.org/10.1007/978-3-642-16161-2_27>. ISBN 978-3-642-16161-2.

[19] Yaghmour, K. *Embedded Android: Porting, Extending, and Customizing*. Oreilly and Associate Series, O'Reilly Media, Incorporated, 2013. ISBN 9781449308292.

Prílohy

Zoznam príloh

Príloha A

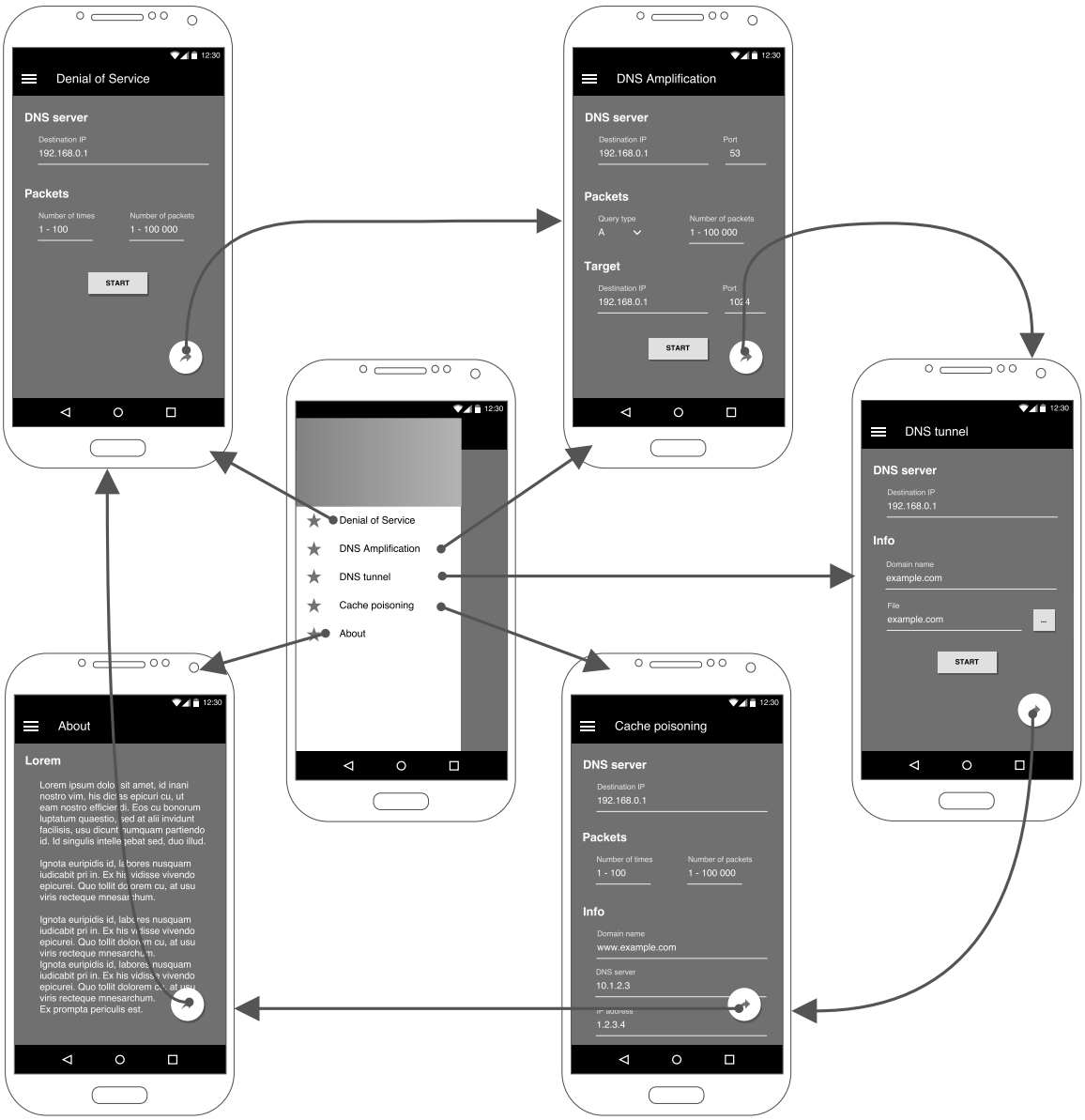
Obsah CD

Priložené CD obsahuje:

- `src` – adresár obsahujúci zdrojové súbory aplikácie
- `tex` – adresár obsahujúci zdrojové LaTeX súbory tejto práce a návodu
- `BP.pdf` – text tejto práce
- `manual.pdf` – návod k aplikácii
- `dnsattacks.apk` – inštalačný súbor aplikácie

Príloha B

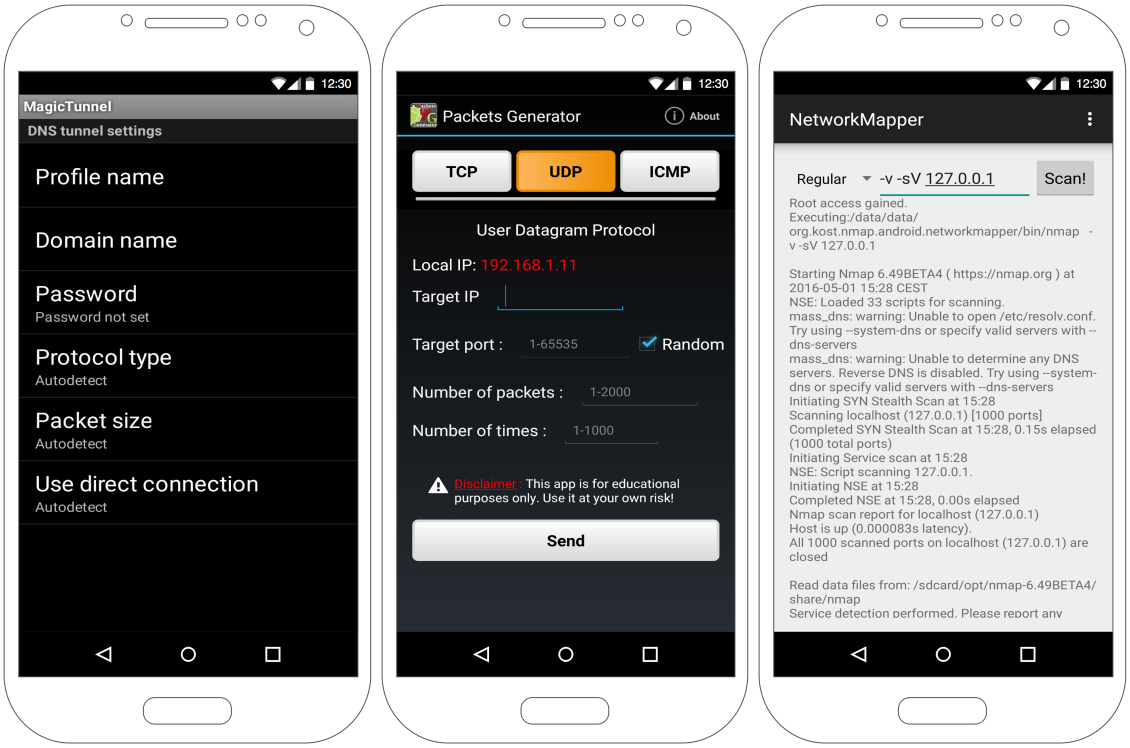
Mockup



Obr. B.1: Návrh obsahu a ovládania aplikácie

Príloha C

Ukážky podobných aplikácií



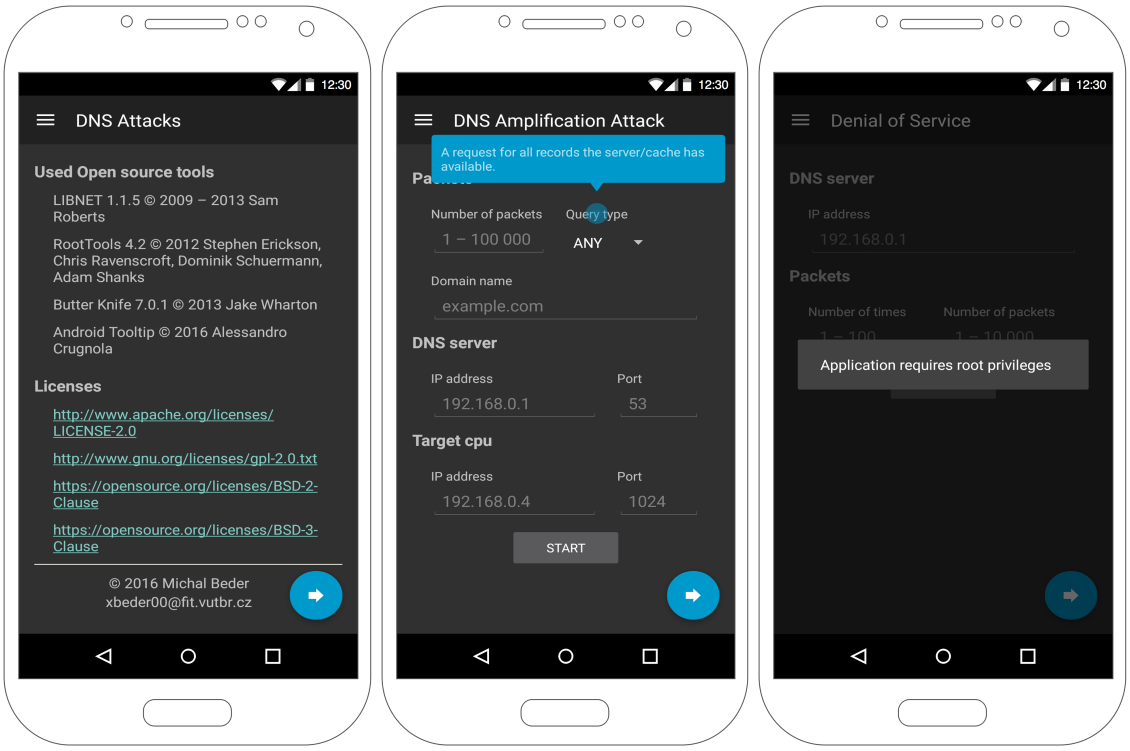
(a) *Magic Tunnel* (b) *Packets Generator* (c) *Network Mapper*

Obr. C.1: Aplikácie podobného zamerania

Príloha D

Návod na inštaláciu a ovládanie aplikácie

Inštalácia aplikácie *DNS Attacks* sa nijako nelíši od inštalácie iných aplikácií. Na priloženom CD sa nachádza súbor `dnsattacks.apk`. Ten stačí skopírovať na cieľové zariadenie a po jeho otvorení prebehne inštalácia. Dôležitá je dostupnosť `root` oprávnení na cieľovom zariadení. Ak tieto oprávnenia nie sú dostupné, zobrazí sa po spustení aplikácie varovanie uvedené na obrázku D.1c. Aplikácia je potom takto zablokovaná a nie je možné ju bežným spôsobom používať.



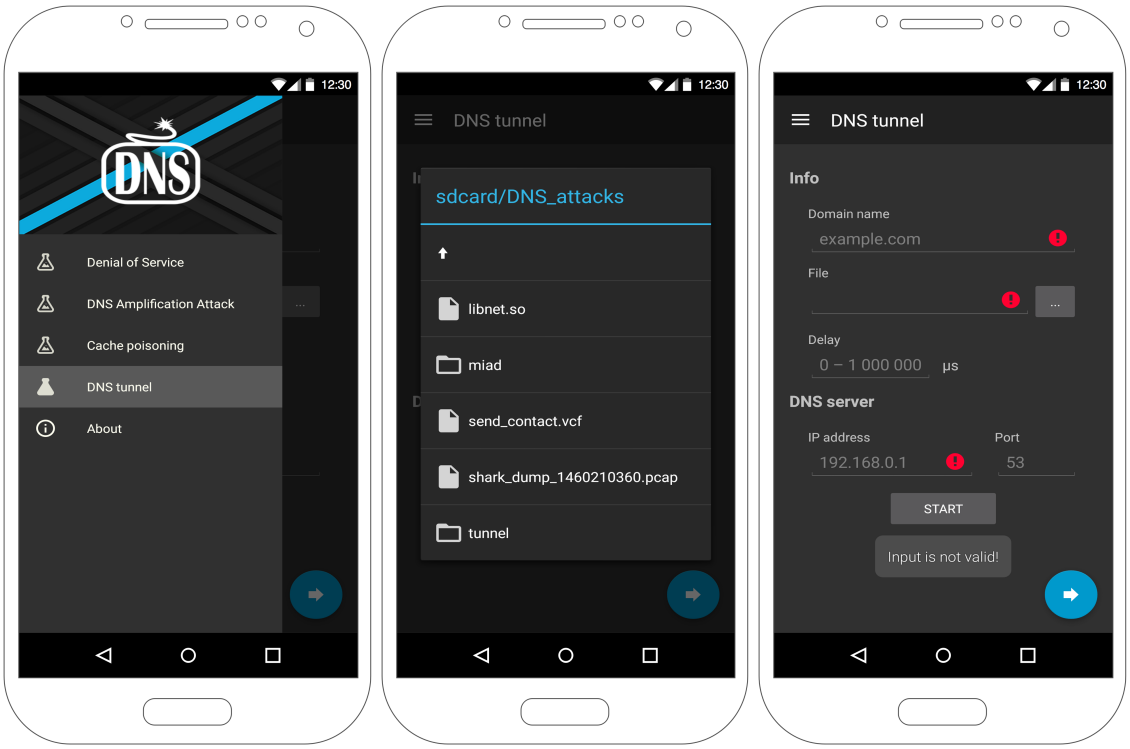
(a) O aplikáci (b) Nápoveda (c) Chýbajúce `root` oprávnenia

Obr. D.1: Ukážky aplikácie *DNS Attacks*

Po úspešnej inštalácii a spustení aplikácie je možné si vybrať jeden z podporovaných typov útokov. Gestom potiahnutia po obrazovke smerom doprava sa objaví bočný navigačný panel ako na obrázku D.2a. Prepínať medzi jednotlivými obrazovkami je možné aj tlačidlom, ktoré je vidieť na každom obrázku v pravom dolnom rohu.

Po zvolení typu útoku je potrebné vyplniť požadované informácie. Okrem vyplnenia textových polí môže byť vyžadovaná napríklad voľba súboru ako na obrázku D.2b. V dialógu výberu sú graficky odlíšené adresáre a súbory. Je tak jednoduché nájsť hľadaný súbor.

V prípade, že povinné vstupy nie sú vyplnené, aplikácia ich po stlačení tlačidla START zvýrazní a zobrazí „Toast“ správu, ako je vidieť na obrázku D.2c. Po kliknutí na niektoré popisné texty sa zobrazí nápoveda s podrobnejším popisom. Príklad je uvedený na obrázku D.1b.



(a) Navigačný panel (b) Výber súboru (c) Povinné vstupy

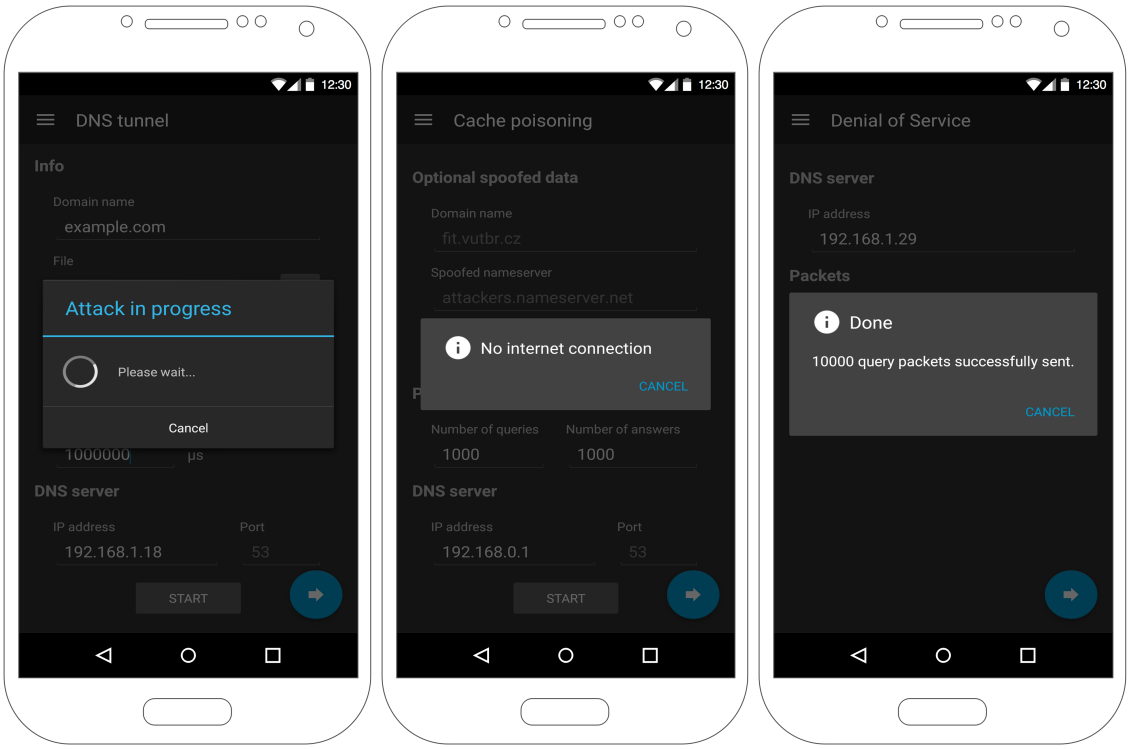
Obr. D.2: Výber útoku z zadávanie vstupov

Všetky použité knižnice tretích strán sú spomenuté priamo v informáciách o aplikácii (posledná položka v navigačnom paneli na obrázku D.2a). Spolu s nimi sú uvedené aj licencie, pod ktorými boli vydané. Ukážka informácií o aplikácii je uvedená na obrázku D.1a.

Po správnom zadaní požadovaných vstupov môžu nastať tieto situácie:

- Nedostupné pripojenie – v prípade, že nie je sieťové pripojenie dostupné pred začatím útoku, upozorní na to dialóg uvedený na obrázku D.3b. V prípade, že pripojenie vypadne počas prebiehajúceho útoku, aplikácia po jeho skončení bude informovať koľko paketov sa nepodarilo odoslať.
- Zahájenie generovania útoku – pokiaľ je sieťové pripojenie dostupné, zaháji sa útok. Aplikácia o tom informuje prostredníctvom dialógu uvedenom na obrázku D.3a.

Pokiaľ prebiehajúci útok nie je prerušený tlačidlom CANCEL, zobrazí sa po jeho skončení informačný dialóg s uvedeným počtom úspešne alebo neúspešne odoslaných paketov. Príklad, ako môže takýto dialóg vyzerať je vidieť na obrázku D.3c. Tento konkrétny dialóg sa zobrazí po úspešnom odoslaní 10 000 paketov po skončení DoS útoku.



(a) Prebiehajúci útok (b) Nedostupná sieť (c) Odoslané pakety

Obr. D.3: Stavy aplikácie po spustení útoku tlačidlom START