Elektrotehnički fakultet Univerziteta u Beogradu

Reverzni inženjering Android aplikacija

Diplomski rad

Mentor
Prof. dr Boško Nikolić

Student
Aleksandar Abu-Samra, 252/08

Beograd, septembar 2013.

Sažetak

Ovaj rad je nastao kao predmet istraživanja tehnika reverznog inženjeringa nad aplikacijama za Android platformu. Opisana je struktura platforme, izvršnih (APK) fajlova, testirani su alati za dekompajliranje, analiziran kod, opisane pretnje i predložene metode zaštite.

Ključne reči: Android, Java, mobilne aplikacije, APK, reverzni inženjering, diplomski rad

1. Sadržaj

2. Uvod

Kao rezultat eksplozije tržišta pametnih telefona, svedoci smo rastuće potrebe za mobilnošću i produktivnošću na svakom koraku. Da bi zadovoljili te potrebe i ostali kompetentni, softverske firme i pojedinci jako brzo menjaju tehnologije, alate, uče nove programske jezike i platforme. Pritisak tržišta koji diktira kratke rokove definitivno ostavlja jake posledice na kvalitet softverskih proizvoda, a pri određivanja prioriteta, testiranje i zaštita redovno odlaze na dno lestvice.
U scenarijima kada telefone koristimo za privatne i poslovne mejlove, društvene mreže i elektronsko poslovanje, poslednje što želimo je da neko drugi ima pristup našim podacima, a upravo to je ono što se dešava na ovom mladom tržištu. Pogođene su sve platforme, a Android nikako nije izuzetak. Postoji zastrašujući broj metoda kojima se mogu poslužiti ”loši momci” u krađi naših podataka, a reverzni inženjering je samo jedna od njih.

2.1. Definicija reverznog inženjeringa

Reverzni inženjering je proces analiziranja, rastavljanja i otkrivanja tehničkih karakteristika proizvoda ili servisa sa ciljem da se otkrije način na koji funkcioniše, često da bi se kasnije modifikovao, kopirao ili napravio sličan.
Reverzni inženjering u svetu softvera je proces otkrivanja izvornog koda aplikacija iz izvršnog fajla, a najčešće se analizira u bajtkodu.

2.2. Motivacija

Motivacija za pisanje rada na temu reverznog inženjeringa je proistekla iz želje za boljim razumevanjem metoda zaštite na Android platformi. U procesu programiranja se često gubi iz vida šira slika, a fokus se baca samo na direktni proizvod. Svest inženjera se koncentriše na kreativni proces i viziju proizvoda, što je i logično, ali tek razmišljanjem “unazad” mogu se uvideti ozbiljni propusti u dizajnu softvera.

2.3. Metodologija rada

Ovaj rad je sastavljen uz pomoć dva načina informisanja:

- Istraživanjem pisanih saznanja o svim relevantnim temama, u svrhu dobijanje šire slike i boljeg razumevanja celokupnog koncepta.
- Primenom i posmatranjem delovanja istraženih tehnika nad Android aplikacijom specijalno kreiranom za potrebe rada.

Kombinovanjem ova dva pristupa dobijen je rad koji iz prve ruke govori o problematikama ovakve teme, ali takođe daje i opšte informacije i smernice čitaocu u situacijama kada bi podrobno istraživanje izlazilo van okvira ovakvog rada.

2.4. Pregled stanja u relevantnoj literaturi

Kako je Android platforma relativno mlada, tema reverznog inženjeringa nad ovim sistemom i dalje se smatra egzotikom, bar kada su u pitanju veliki izdavači i akademski krugovi. Literature, naravno, ipak ima, ali se lakše nalazi u blog arhivama posvećenih individualaca, javno dokumentovanim praksama specijalizovanih firmi ili manjim tezama radoznalog univerzitetskog osoblja.

Tekstovi dobijeni na ovakav način, iako jako informativni, često se bave samo specijalnim slučajevima i uskim segmentima problema, a početnici nemaju alternativu osim da procesiraju sve informacije odjednom i sami odvajaju bitne od nebitnih, otežavajući i usporavajući sebi proces učenja.

Ovaj rad pokušava da reši taj problem, polazeći od neophodnog predznanja ali brzim uvodom u konkretne metode, njihov prikaz kroz primere i objašnjenju o načinima zaštite.

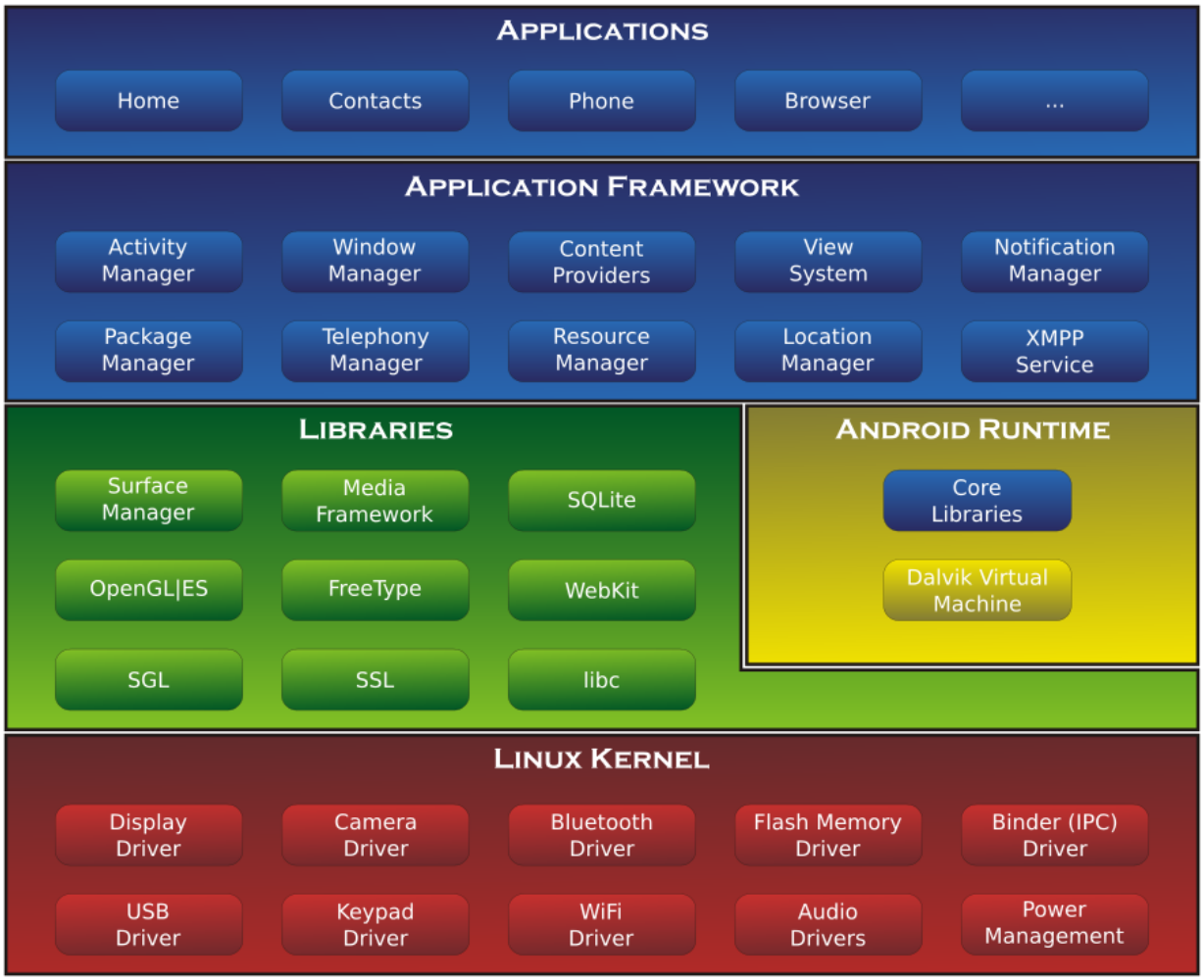
2.5. Ciljevi

Glavni cilj ovog rada je pronalaženje dobrih inženjerskih praksi na polju zaštite Android softvera. Kroz rad će to biti realizovano boljim razumevanjem tehnika reverznog inženjeringa, predstavljanjem realnih scenarija i konkretnih alata.

Za Android programere koji žele da steknu osnovnu sliku o sigurnosnim propustima i dobrim praksama, ovaj rad može biti dovoljan. Ipak, za nekoga ko želi dublje istraživati metode reverznog inženjeringa, ovo je samo početna stanica u svet obrnutog razmišljanja.

3. Android platforma

Android je operativni sistem za pametne telefone i tablete. Lansiran kasne 2007. godine, za razliku od ostalih mobilnih operativnih sistema, od početka je bio dostupan različitim proizvođačima. Google, čije je Android vlasništvo, je postavio jasne ciljeve još na početku razvoja: želeo je odličnu integraciju sa svojim servisima, sinhronizaciju i multi-tasking, što se kasnije ispostavilo kao dobra odluka.
Popularnost Android uređaja je brzo rasla i po podacima iz maja 2013. ova platforma je postala ubedljivi lider na tržištu, kako na pametnim telefonima tako i na tabletima.



Slika 3.1. - Arhitektura Android platforme.

3.1. Kako Android sistem funkcioniše

Iako se Android aplikacije programiraju u Javi, SDK (software development kit), koji je se može javno preuzeti, pretvara svaki Java kod u Android izvršni paket (APK). Ovi APK fajlovi izvršavaju se nad Dalvik virtuelnoj mašini (DVM) napravljenoj isključivo za potrebe Android sistema.

Android aplikacije interaguju sa sistemom ali i među sobom raznim međuprocesnim komunikacijama. Svaka komponenta APK fajla igra drugačiju ulogu u ponašanju aplikacije i može biti aktivirana posebno.

3.1.1. Osnovne komponente Android aplikacije

Activity

Aktivnosti su pojedinačni prikazi prozora sa korisničkim interfejsom. Iako funkcionišu zajedno u jednom koherenhnom korisničkom iskustvu, one su nezavisne jedna od druge. Na primer, jedna aktivnost može prikazivati listu novih poruka, dok druga kreira poruku a treća prikazuje individualne poruke.

Service

Servisi su zaslužni za operacije koje se izvršavaju duže vremena, rade kao izolovani procesi i mogu biti startovani iz aktivnosti, ali ne zahtevaju korisnički interfejs. Na primer, servis može puštati muziku u pozadini dok korisnik koristi ekran za kreiranje tekstualne poruke.

Content Provider

Provajderi sadržaja manipulišu setovima podataka koji su deljeni između aplikacija i ponašaju se kao “mostovi” do drugih komponenti aplikacije. Na primer, mnoge aplikacije za pisanje beleški koriste provajdere sadržaja za njihovo čuvanje.

Broadcast receiver

Broadcast prijemnici reaguju na sistemske broadcast objave. Na primer, oni obaveštavaju uređaj da se ekran ugasio ili je baterija na izmaku. Iako ne prikazuju korisnički interfejs, broadcast prijemnici mogu da kreiraju notifikacije u status baru.

3.1.2. Ostale specifičnosti

AndroidManifest.xml

Sve Android aplikacije u root direktorijumu moraju imati Manifest fajl koji obezbeđuje sistemu sve potrebne informacije za rad. Neke od informacija su ime aplikacije, ime paketa, verzija, minimalni API na kojem se izvršava, komponente, neophodne dozvole, potrebne biblioteke...

3.2. Proces izgradnje Android aplikacije

3.2.1. Pred-kompajliranje resursa

Prvi korak u procesu kompajliranja aplikacije uključuje generisanje Java izvornih fajlova iz uključenih Android resursa. Ti resursi, koji se nalaze u *res/* direktorijumu projekta, sadrže fajlove kao što su ikonice, informacije o izgledu aplikacije, stringovi itd. Oni se kompajliraju koristeći *aapt* alat u fajl nazvan R.java, koji se skladišti u *gen/* direktorijumu. Ako se zaviri u generisani fajl, mogu se videti definicije mnogih konstanti:

```
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

Konstante se koriste kao pokazivači na resurse, koji su skladišteni u paketnom fajlu u kasnijem koraku.

3.2.2. Pred-kompajliranje Service interfejsa

Drugi korak takođe podrazumeva generaciju Java koda. Ukoliko projekat koristi bilo kakav Service interfejs, potrebno je uključiti definicije interfejsa (fajlovi .aidl ekstenzije) u projekat. Oni nalikuju običnim Java interfejsima:

```
interface IsimpleService {
    String echo(in String s);
}
```

Aidl alat se koristi za generisanje pravih Java interfejsa za ove servise. Java izvorni fajlovi će imati isto ime kao ulazni fajlovi i biće kreirani u *gen/* direktorijumu. Ovako generisani izvorni fajlovi služe kao baza za implementiranje poziva servisnih interfejsa u kodu.

3.2.3. Java kompajliranje

U ovom koraku je kompletan Java kod generisan i spreman za kompajliranje. Vrši se standardna procedura kompajliranja iz .java izvornih fajlova (i napisanih i generisanih) u .class bajtkod fajlove. Binarni bajtkod fajlovi se smeštaju u *bin/classes* direktorijum.

3.2.4. Prevođenje bajtkoda

Nakon kompajliranja, dobija se standardni Java bajtkod koji bi se mogao pokrenuti na standardnoj Java virtuelnoj mašini. Ipak, Android koristi Dalvik virtuelnu mašinu koja zahteva drugačiji bajtkod format. Iz tog razloga, nakon kompajliranja, koristi se *dx* alat za prevod class fajlova u Dalvik izvršne fajlove (DEX). Ovaj postupak obuhvata i sve class fajlove skladištene u eksternim JAR bibliotekama koje su dodate u projekat. Sve klase su potom upakovane u jedinstveni izlazni fajl, *classes.dex*, koji se smešta u *bin/* direktorijum.

3.2.5. Prepakivanje resursa

Dalje, resursi se ugrađuju u Android paketni fajl, ponovo uz pomoć *aapt* alata. Kreira se resursni paket, imenovan kao i aplikacija sa *ap_* sufiksom u *bin/* direktorijumu. XML fajlovi se konvertuju u odgovarajuće binarne datoteke, a stringovi u datoteku *resources.arsc*.

3.2.6. Debagovanje i potpisivanje paketa

Sada su sve komponente spremne i čekaju da budu upakovane u jedinstven APK fajl sa imenom aplikacije. Ovaj korak podrazumeva i potpisivanje paketa sa debug ili release ključem. Android pakete sastavlja *apkbuidler* alat, koji preuzima podatke sa svih do sada pomenutih lokacija i smešta paket u *bin/* direktorijum, imenovan sa *MyApp-debug-unaligned.apk.*

3.2.7. Poravnavanje na 4-bitne reči

Kao poslednji optimizacioni korak, resursni fajlovi u paketu poravnava na 4-bitne reči koristeći *zipalign* alat. Ovo omogućava Dalvik virtuelnoj mašini da memorijski mapira delove fajla za efikasniji pristup. Kao ulaz se koristi prethodno dobijen, neporavnat APK - *MyApp-debug-unaligned.apk*, da bi se proizveo fajl *MyApp-debug.apk*. Dobijen je finalni, potpisan i poravnat Android izvršni paket, spreman za instaliranje na Android uređaje.

4. Analiza Android koda

4.1. Pregled alata za analizu koda

4.1.1. Dexdump

Dexdump je alat koji dolazi u okviru Android SDK paketa i oficijalni je alat za prikazivanje Dalvik bajtkoda iz DEX fajlova. Alat je daleko od idealnog i njegov izlaz je jako teško čitati, pošto ne deli kod na manje delove već ga ispisuje u celosti.

4.1.2. Dedexer

Dedexer je alternativa alatu dexdump. On je takođe disasembler DEX fajlova, ali svoj izlaz pakuje u manje DDX fajlove čime poboljšava čitljivost.

4.1.3. Apktool

Apktool je alat za reverzni inženjering zatvorenih, binarnih fajlova Android aplikacija. On može dekodovati resurse do skoro originalne forme i ponovo ih izgraditi nakon modifikovanja. Koristi se i za dekodovanje *smali* koda korak po korak. Lak je za korišćenje pri radu sa aplikacijama zbog strukture fajlova koja je slična projektnoj, ali i zbog automatizacije repetitivnih zadataka kao što je sklapanje APK fajla i slično.

4.1.4. Dex2jar

Ovaj alat otvorenog koda se sastoji iz četiri komponente:

- *dex-reader* - koji čita Dalvik izvršne fajlove (.dex/.odex)
- *dex-translator* - zadužen za konvertovanje; on čita dex instrukcije u *dex-ir* formatu, posle čega ih optimizuje i konvertuje u ASM format
- *dex-ir* - koristi ga dex-translator za representaciju dex instrukcija
- *dex-tools* - alat za rad sa .class fajlovima

4.1.5. Smali / backsmali

Smali/baksmali je asembler/disasembler za dex format koji koristi Dalvik virtuelnu mašinu. Podržava punu funkcionalnost dex formata (anotacije, informacije o debagovanju, linijama, itd.).

4.1.6. JD-GUI

JD-GUI je samostalna grafička aplikacija koja prikazuje Java kod iza .class fajlova. Iz njega se može rekonstruisati izvorni kod i direktno pristupiti metodama i poljima klase.

4.1.7. Androguard

Androguard je tekstualni alat napravljen u Pythonu koji po svojim funkcionalnostima predstavlja alternativu *apktool*-u, ali sa opcijom ugrađivanja skripti koje ga proširuju i optimizuju rad.

4.1.8. Ostali alati

Postoji još mnoštvo alata za razne platforme i namene, komercijalni i besplatni i iza kojih stoje pojedinci ili kompanije, ali oni neće biti pominjani u nastavku ovog rada. Bitno je napomenuti da oni postoje i da nijedan nije 100% savršen, pa se često dešava da, iako pokrivaju iste funkcionalnosti, neki alati, za neke konkretne slučajeve jednostavno ne rade, pa treba pokušavati i eksperimentisati sa drugima.

4.2. Proces dekompajliranja

4.2.1. Otpakivanje APK fajla

Nakon otpakivanja APK fajla (unzip operacija) već se mogu videti svi multimedijalni resursi koje aplikacija koristi, kao što su slike i audio i video zapisi. Ipak, kompletna programska logika i grafički interfejs aplikacije su sakriveni u binarnim datotekama *classes.dex* i *AndroidManifest.xml*. Da bi se one mogle čitati ili modifikovati, potrebno je koristiti neki od navedenih alata. Koristeći na primer apktool i decompile opciju, odmah dobijamo čitljiv *AndroidManifest.xml* fajl, dok *classes.dex* možemo čitati na nekoliko načina.

4.2.2. Dexdump

Primer korišćenja dexdump alata:

```
# dexdump HelloReverse-debug-unaligned.apk | less
```

4.2.3. Dekompajliranje u smali kod

Primer korišćenja apktool alata:

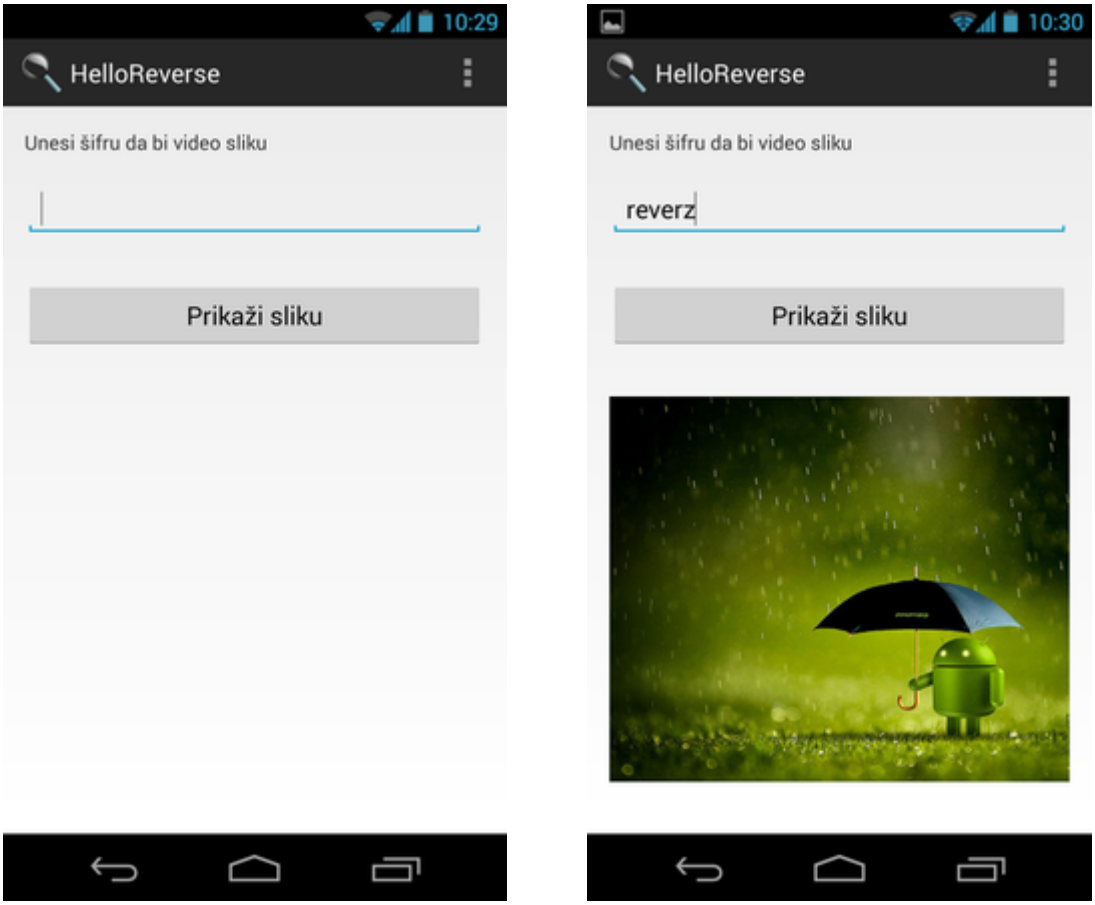
```
# apktool d HelloReverse-debug-unaligned.apk
```

4.2.4. Dekompajliranje u Java kod

Pri dekompajliranju na Java kod se najčešće prvo koristi alat dex2jar, pri čemu se iz .dex fajlovi krairaju fajlovi .jar formata (standardna Java arhiva) nad kojima se kasnije primenjuju tehnike klasičnog dekompajliranja Java koda. Iako Java dekompajleri postoje već duže vreme, njihovi rezultati nisu uvek tačni i dovode do neispravnog koda. Sa druge strane, programerima je neuporedivo prirodnije da razumeju Java kod umesto smali-ja, pa se često prave kompromisi.

4.3. Primer na realnoj aplikaciji

Kako je legalnost postupka reverznog inženjeringa nad vlasničkim aplikacijama diskutabilna, za potrebe ovog rada napravljena je demo aplikacija isprojektovana tako da prikaže osnovni skup slabosti koje se pominju.
Ideja aplikacije je da predstavi jako lošu praksu ugrađivanja (hardkodovanja) promenljivih za koje želimo da budu tajne. Takođe, iz primera će se jasno videti lakoća otkrivanja resursnih fajlova (kao što su slike, audio i video), kao i metode zaobilaženja ograničenja iako ne možemo saznati lozinku.



Slike 4.1. i 4.2. - Ekran test aplikacije, pre i nakon unosa tajnog stringa.

Aplikacija poseduje jedan prozor (*MainActivity*) na koji je poređano nekoliko vizuelnih elementa:

- *Label*, koji povlači svoju tekstualnu vrednost iz fajla *strings.xml*
- *EditText*, koji služi za korisnički unos tajnog stringa
- *Button*, koji proverava da li je tajni string ispravan i prikazuje sakrivenu sliku ako jeste
- *ImageView* - sakrivena slika

Nakon unosa teksta i pritiska na dugme, aplikacija proverava string iz *EditText* polja sa hardkodovanom string promenljivom iz klase *MainActivity.java*. U slučaju da su isti - prikazuje se slika.

4.3.1. Otpakivanje APK fajla i vađenje resursa

Hajde da na realnom primeru pogledamo šta se sve može saznati jednostavnim otpakivanjem (unzip-ovanjem) APK fajla koji je napravljen:

```
./AndroidManifest.xml
./classes.dex
./META-INF
./res
./resources.arsc
./META-INF/CERT.RSA
./META-INF/CERT.SF
./META-INF/MANIFEST.MF
./res/drawable-hdpi
./res/drawable-mdpi
./res/drawable-xhdpi
./res/drawable-xxhdpi
./res/layout
./res/menu
./res/drawable-hdpi/android_firewall.jpg
./res/drawable-hdpi/ic_launcher.png
./res/drawable-mdpi/ic_launcher.png
./res/drawable-xhdpi/ic_launcher.png
./res/drawable-xxhdpi/ic_launcher.png
./res/drawable-xxhdpi/img.jpg
./res/layout/activity_main.xml
./res/menu/main.xml
```

Ako je cilj bio samo da se dođe do sakrivene slike, to je već postignuto. Prolaskom kroz *./res/drawable-** direktorijume može se videti slika *android_firewall.jpg* u punoj rezoluciji, koju bez daljih obrada možemo koristiti i kopirati za sopstvene potrebe, iako je to bila "sakrivena" opcija u aplikaciji.
Takođe, može se videti i fajl sa ikonicom aplikacije, i potencijalno bilo koji drugi multimedijalni resurs koji je ugrađen u APK fajl.

4.3.2. Dekompajliranje u smali - apktool

Za razumevanje koda aplikacije, mora se pribegnuti posebnim alatima. Koristeći *apktool* jako lako se može dobiti lista klasa u *smali* kodu i čitljiv *AndroidManifest.xml* fajl. Evo dobijenog izlaza:

```
# apktool d HelloReverse-debug-unaligned.apk
I: Baksmaling...
I: Loading resource table...
I: Loaded.
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /apktool/framework/1.apk
I: Loaded.
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Done.
I: Copying assets and libs...
```

Dobijene klase:

```
./smali/com/dipl/helloreverse/BuildConfig.smali
./smali/com/dipl/helloreverse/MainActivity$1.smali
./smali/com/dipl/helloreverse/MainActivity.smali
./smali/com/dipl/helloreverse/R$attr.smali
./smali/com/dipl/helloreverse/R$dimen.smali
./smali/com/dipl/helloreverse/R$drawable.smali
./smali/com/dipl/helloreverse/R$id.smali
./smali/com/dipl/helloreverse/R$layout.smali
./smali/com/dipl/helloreverse/R$menu.smali
./smali/com/dipl/helloreverse/R.smali
./smali/com/dipl/helloreverse/R$string.smali
./smali/com/dipl/helloreverse/R$style.smali
```

Otvaranjem fajlova *MainActivity.smali* i *MainActivity$1.smali* mogu se uočiti značajne informacije o logici *MainActivity* klase, što je biti prikazano poređenjem ekvivalentnih segmenata smali i Java koda:

Hardkodovane promenljive su jasno čitljive

Smali:

```
const/16 v0, 0x7dd
iput v0, p0, Lcom/dipl/helloreverse/MainActivity;->hcInt:I

const-wide v0, 0x401d28f5c28f5c29L
iput-wide v0, p0, Lcom/dipl/helloreverse/MainActivity;->hcDouble:D

const-string v0, "reverz"
iput-object v0, p0,
Lcom/dipl/helloreverse/MainActivity;->hcString:Ljava/lang/String;
```

Java:

```
public int hcInt = 2013;
protected double hcDouble = 7.29;
private String hcString = "reverz";
```

Primetiti da su sve promenljive vidljive, bez obzira na kontrolu pristupa na Javi.

Praćenje definisanja metoda

Smali:

```
.method protected onCreate(Landroid/os/Bundle;)V
    .locals 1
    .parameter "savedInstanceState"

    .prologue
    invoke-super {p0, p1},
Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V

    …

    return-void
.end method
```

Java:

```
protected void onCreate(Bundle savedInstanceState) { … }
```

Dodavanje Listener-a na dugme

Smali:

```
iget-object v0, p0,
Lcom/dipl/helloreverse/MainActivity;->button:Landroid/widget/Button;

new-instance v1, Lcom/dipl/helloreverse/MainActivity$1;

invoke-direct {v1, p0},
Lcom/dipl/helloreverse/MainActivity$1;-><init>(Lcom/dipl/helloreverse/MainActivi
ty;)V

invoke-virtual {v0, v1},
Landroid/widget/Button;->setOnClickListener(Landroid/view/View$OnClickListener;)
V
```

Java:

```
button.setOnClickListener(new View.OnClickListener() { … }
```

Deo koda zadužen za prikaz skrivene slike

Smali:

```
.method public onClick(Landroid/view/View;)V
    .locals 2
    .parameter "arg0"

    .prologue
    iget-object v0, p0,
Lcom/dipl/helloreverse/MainActivity$1;->this$0:Lcom/dipl/helloreverse/MainActivi
ty;

    #getter for:
Lcom/dipl/helloreverse/MainActivity;->hcString:Ljava/lang/String;
    invoke-static {v0},
Lcom/dipl/helloreverse/MainActivity;->access$000(Lcom/dipl/helloreverse/MainActi
vity;)Ljava/lang/String;
```

```
    move-result-object v0

    iget-object v1, p0,
Lcom/dipl/helloreverse/MainActivity$1;->this$0:Lcom/dipl/helloreverse/MainActivi
ty;

    iget-object v1, v1,
Lcom/dipl/helloreverse/MainActivity;->editText:Landroid/widget/EditText;

    invoke-virtual {v1},
Landroid/widget/EditText;->getText()Landroid/text/Editable;

    move-result-object v1
    invoke-virtual {v1}, Ljava/lang/Object;->toString()Ljava/lang/String;

    move-result-object v1
    invoke-virtual {v0, v1}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

    move-result v0
    if-eqz v0, :cond_0

    iget-object v0, p0,
Lcom/dipl/helloreverse/MainActivity$1;->this$0:Lcom/dipl/helloreverse/MainActivi
ty;

    iget-object v0, v0,
Lcom/dipl/helloreverse/MainActivity;->image:Landroid/widget/ImageView;

    const/high16 v1, 0x7f02
    invoke-virtual {v0, v1}, Landroid/widget/ImageView;->setImageResource(I)V

    :cond_0
    return-void
.end method
```

Java:

```
public void onClick(View arg0) {
    if (hcString.equals(editText.getText().toString())) {
        image.setImageResource(R.drawable.android_firewall);
    }
}
```

Modifikovanje .smali fajlova i krekovanje aplikacije

Ovako dobijen kod Android aplikacije se može jednostavno iskoristiti ili modifikovati u cilju otkrivanja skrivenih opcija i kontrole pristupa. Evo nekoliko ideja šta se može uraditi na konkretnom primeru:

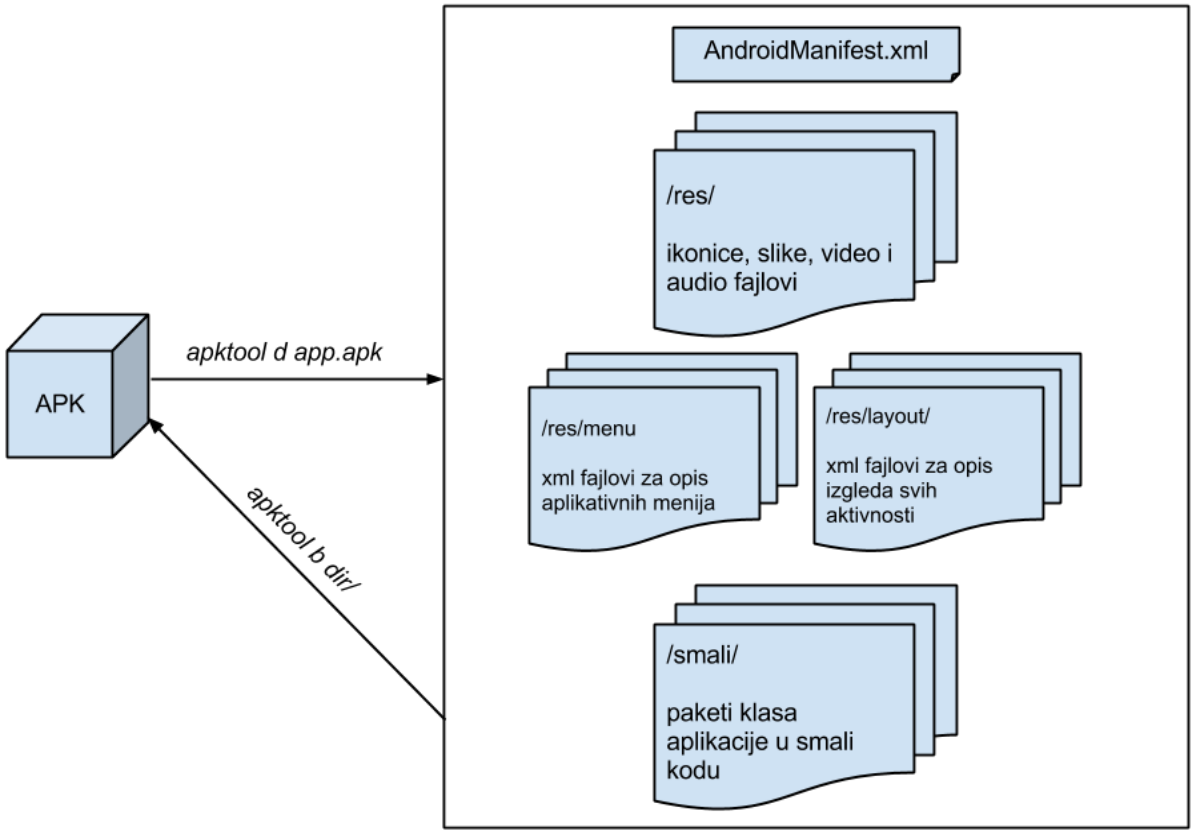
- Sadržaj string promenljive *hcString*, koji se kasnije koristi u proveri tačnosti lozinke možemo lako pročitati i iskoristiti u originalnoj aplikaciji da bi nam se prikazala sakrivena slika.
- Moguće je promeniti sadržaj string promenljive *hcString* u bilo koji drugi i koristiti njega za prikaz sakrivene slike, naravno, u modifikovanoj aplikaciji, nakon reasemblovanja .smali fajlova nazad u APK.
- U delu *smali* koda gde se poredi jednakorst tajnog stringa sa unetim, uslovni skok (`if-eqz`) se može zameniti bezuslovnim (`goto`) ili nekim drugim skokom.

- Ili se mogu kompletno ukloniti uslovni skokovi i ostaviti samo deo koda koji učitava sliku `(invoke-virtual {v0, v1}, Landroid/widget/ImageView;->setImageResource(I)V)`

Kao što se može primetiti, načina da se zaobiđu ograničenja originalnog koda ima mnogo, i svakako se ne treba ograničiti samo na predložene. Potrebno je da programeri budu svesni ovakvih mogućnosti i na vreme razmišljaju o dizajnu svog softverskog rešenja.

Reasemblovanje .smali fajlova u APK

Koristeći *apktool*, dekompalirana i modifikovana aplikacija se može ponovo sastaviti u funkcionalan APK fajl jednostavnom *build* komandom. Nakon toga, treba još samo potpisati fajl alatom kao što je *Jarsigner* u kombinaciji sa *Keytool-om* (oba su sastavni delovi JDK paketa).



Slika 4.3. - Dijagram toka dekompajliranja sa apktool-om.

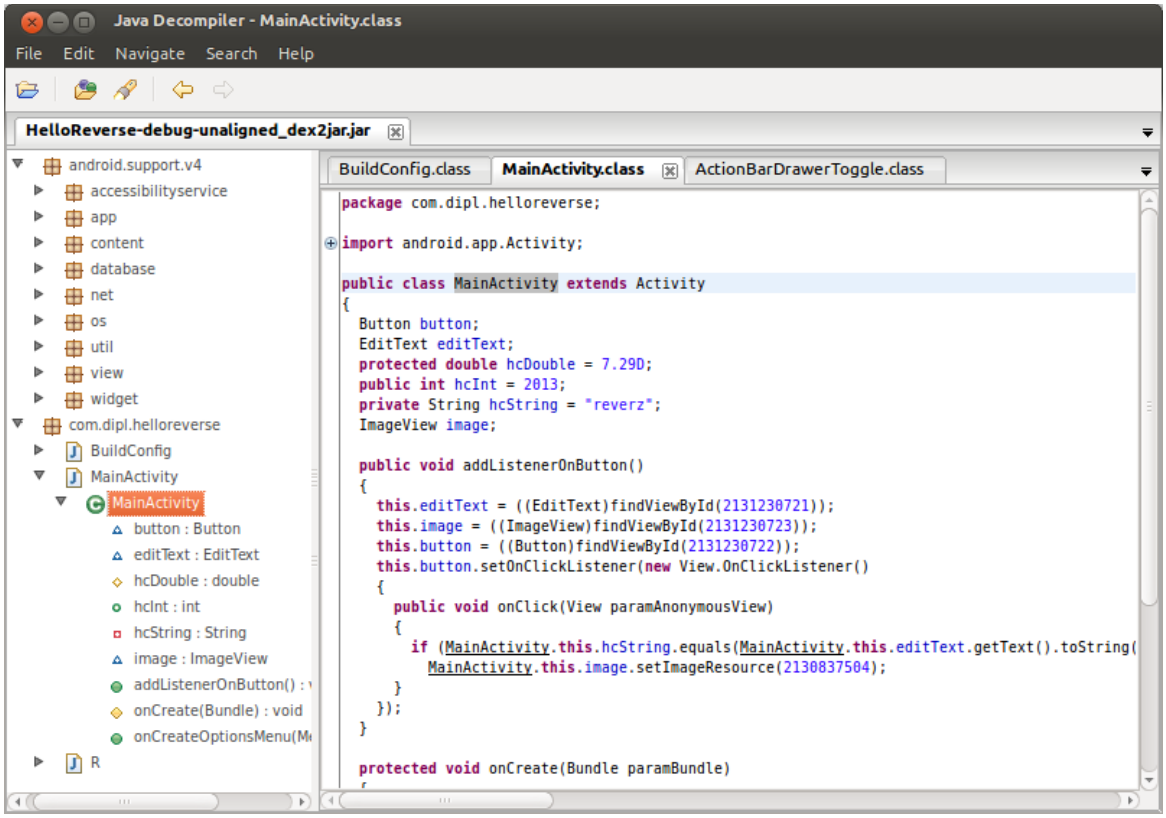
4.3.3. Dekompajliranje u Java kod - dex2jar i JD-GUI

Konvertovanje APK fajla u JAR - dex2jar

Jednostavnim prosleđivanjem putanje APK fajla kao parametra programa dex2jar, automatski se kreira ekvivalentni JAR fajl koji se nadalje može dekompajlirati standardnim Java dekompajlerima.

Čitanje klasa iz .jar fajla - JD-GUI

Iz JD-GUI alata, otvaranjem JAR fajla koji smo kreirali, možemo jako uredno videti dekompajlirane verzije svih klasa koje se sadržane u APK fajlu, skoro kao što bi smo to videli u nekom Java razvojnom okruženju.

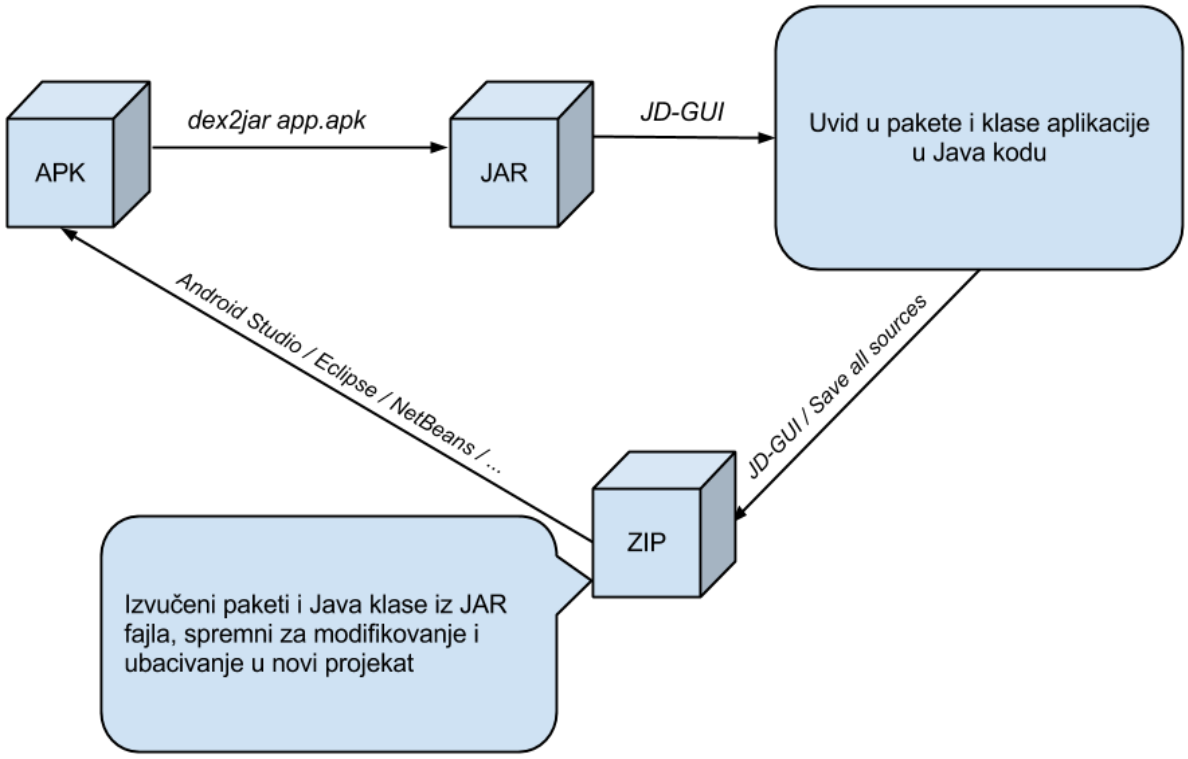


Slika 4.4. - JD-GUI prikaz dekompajliranog koda.

Iako ne 100% pouzdan, ovakav pregled koda, samim tim što je u Java programskom jeziku, nam daje značajno jasniji uvid u logiku aplikacije, bolji pregled klasa i metoda i uočljivije promenljive. Sve klase se mogu lako eksportovati iz JD-GUI programa u neko razvojno okruženje poput IntelliJ IDEA ili Eclipse i tamo dalje modifikovati.

Rekompajliranje .java fajlova u Android aplikaciju

Bitno je napomenuti da reakompajliranje ovako modifikovanih klasa nazad u funkcionalnu Android aplikaciju može biti značajno otežano. Java dekompajeri nisu savršeni, prave greške i često imaju problem sa nedefinisanim promenljivima, pristupnim opsezima, ili jednostavno ne povezuju zavisnosti između klasa u funkcionalnu celinu. Pažljivom analizom i ručnim prepravljanjima su ovi problemi često rešivi, ali univerzalnih postupaka nema pa detalji ovakvog načina rekompajliranja izlaze izvan opsega ovog rada.



Slika 4.5. - Dijagram toka dekompajliranja sa dex2jar i JD-GUI alatima.

5. Pretnje

Primeri navedeni u prethodnoj sekciji su samo deo celokupnog problema sigurnosti na Android platformi. Da bi se stekla šira slika, u ovoj sekciji će problemi biti sagledani sa više aspekata.

5.1. Problemi Android sigurnosti

5.1.1. Lako reverzibilni kod

Činjenica da se Android aplikacije pišu u Java programskom jeziku znatno olakšava reverzni inženjering. Java arhitektura zajedno sa Java virtuelnom mašinom je tradicionalno bila pogodno tle za razvoj mnogih alata koji za cilj imaju dekompajliranje izvršnih aplikacija.

5.1.2. Root pristup

Android operativni sistem je baziran na Linuxu koji strogo definiše kontrole pristupa svih aplikacija i korisnika prema sistemu. Root-ovanjem telefona, korisnik dobija administratorski pristup svim fajlovima, servisima i sistemskim aplikacijama, pristup hardveru i slično... Ovakve mogućnosti su pogodne programerima za istraživanje, razvoj i prilagođavanje platforme sebi, ali isto tako otvaraju potencijalne rupe u sistemu i mogu biti opasne ako je korisnik nepažljiv.

5.1.3. Mobilni razvoj je rastući trend

Na brzom, surovom i kompetitivnom IT tržištu razvoj aplikacija za mobilne platforme predstavlja nov i neistražen koncept. Programeri su pod stalnim pritiskom i od njih se zahteva brz rad i poštovanje kratkih rokova. Kako je industrija mlada, oni nemaju dovoljno znanja, nisu uvežbani niti svesni dobrih praksi. Čitava hijerarhija projektnog menadžmenta isključivo žuri da izbaci proizvod, a o bezbednosti se gotovo uvek razmišlja tek kada se nešto loše desi.

5.1.4. Veliki broj lako dostupnih alata

U sekciji 4.1. prikazano je mnoštvo alata za analizu, dekompajliranje i modifikovanje Android koda, a lista svakako nije konačna. Alati su često besplatni, otvorenog koda i dostupni svim platformama, pa je time i baza potencijalnih korisnika automatski veća.

5.1.5. Spora unapređenja platforme na starijim uređajima

Android platforma je otvorena i dinamična, a Google ulaže velike napore u njenu agilnost i sigurnost, brzo reagujući na sve sigurnosne propuste i greške. Ipak, sama otvorenost platforme dozvoljava krajnjim proizvođačima da modifikuju delove sistema i kao takvog da distribuiraju putem svojih uređaja. To im često daje dodatne funkcionalnosti i drugačiji vizuelni identitet, ali takođe usporava proces unapređenja, pa takvi uređaji dobijaju nove verzije sistema sa po nekoliko meseci zakašnjenja, a često i nikad.

5.2. Vrste pretnji

Android platformu pogađaju razne vrste pretnji, a jedan od najvećih sigurnosnih propusta je svakako činjenica se aplikacije lako šalju na Google Play Store, bez strogih sigurnosnih provera od strane autoriteta. To dovodi do prakse da se pojedine maliciozne aplikacije saknconišu tek nakon što budu prijavljene kao rizične, dok je u međuvremenu šteta već načinjena.
U nastavku, naravo, nisu opisane sve pretnje koje pogađaju Android platformu, već samo one koje se mogu izvesti tehnikom reverznog inženjeringa.

5.2.1. Razumevanje koda aplikacije

Svaki pokušaj razumevanja koda iza softverskih rešenja može otkriti neke od tajni i time oštetiti proizvođača. Objavljivanje takvih saznanja predstavlja dodatni rizik, pa je u softverskoj industriji ustanovljena praksa da se svaki delić komercijalno napisanog softvera smatra privatnim vlasništvom i čuva se kao poslovna tajna.

5.2.2. Krađa koda i resursa

Kompanije često koriste tehnike reverznog inženjeringa za špijuniranje konkurencije. Uvid u kod aplikacije im može dati informacije o algoritmima na koje je uloženo napredno istraživanje i programersko vreme.
Takođe, resursi aplikacije, kao što su autorske slike, muzika, zvučni efekti i video materijal, jako lako mogu biti ukradeni i korišćeni u nelegalne svrhe.

5.2.3. Otkrivanje pristupnih šifara baza podataka, servera, mejlova...

Ukoliko aplikacija direktno komunicira sa bazom podataka na internetu, pristupni podaci mogu lako biti otkriveni i upotrebljeni za krađu informacija ili kompromitovanje baze.
Eventualno otkrivanje načina za pristup serveru je još gori scenario, gde posledice neautorizovanog pristupa mogu biti još fatalnije.

Bilo kakvo drugo pristupanje servisima na internetu, kao što su mejlovi, specijalizovani sajtovi, forumi i blogovi, uz hardkodovane pristupne podatke, može olakšati krađu tih podataka od strane napadača.

5.2.4. Neautorizovani pristup API-ju

Komuniciranje sa serverom putem API-ja umesto direknim pristupom serveru i bazama podataka je svakako dobra praksa. Ipak, uvid u kod aplikacije može otkriti neautorizovanim licima privatne API pozive, koji se mogu iskoristiti u neetičke svrhe.
API-ji velikih servisa, kao što su Twitter, Facebook, Flicker i drugih, iako podrobno testirani, nisu 100% sigurni, pošto jednostavnim saznavanjem API ključa, napadač može imati pristup svim operacijama kao i regularni korisnik.

5.2.5. Prepakivanje i prodavanje aplikacije

Skidanjem aplikacije sa Google Play Store-a, dekompajliranjem, modifikacijom imena, potpisa, dizajna ili bilo čega drugog, pa ponovnim kompajliranjem i slanjem na market, moguće je praktično ukrasti kompletan proizvod i prodavati ga nezavisno i bez znanja originalne kompanije.
Drugi način generisanja novca nad tuđom aplikacijom bi bio ubacivanje kodova ka sopstvenim reklamnim servisa, gde bi umesto originalnom autoru, novac odlazio direktno napadaču.

5.2.6. Piraterija

Postupkom sličnim opisanim u prethodnoj stavci, APK fajlovi se mogu modifikovati da zaobiđu ograničenja probnih verzija softvera i tako eliminišu potrebu za kupovinom ili pretplatom na isti. Ovaj postupak se popularno zove “krekovanje”, a aplikacije modifikovane na taj način se lako mogu naći na specijalizovanim piratskim sajtovima, torentima ili forumima.

5.2.7. Maliciozni softver

Pored “etičkog” pristupa krekovanju softvera, postoji i ono “neetičko”, gde se umesto zaobilaženja ograničenja, u kod ubacuju algoritmi sposobni za krađu podataka od korisnika, tajno slanje SMS ili obavljanje poziva ka internacionalnim brojevima, praćenje aktivnosti korisnika, sabotažu određenih funkcija itd. Ovakve aplikacije se jako brzo uklanjaju sa Google Play-a, ali ih korisnik ipak može naći na piratskim sajtovima ili forumima, i iz neznanja instalirati.

5.2.8. Istraživanje sakrivenih opcija

Pretraživanjem koda, mogu se otkriti i sakrivene opcije ili meniji, koji se planiraju tek za neku buduću verziju aplikacije. Nekada to ne mora značiti mnogo, ali postoje situacije u kojima takvo "curenje" informacija može značajno poremetiti tajming plasiranja podataka u javnost.

5.2.9. Otkrivanje kriptografskih metoda

Ukoliko proizvođač softvera koristi posebne kriptografske metode u svojim proizvodima, čiju sigurnost zasniva na tajnosti koda a ne na dobrim kriptografskim praksama, razumevanje tih metoda može ozbiljno naškoditi tajnosti celokupne komunikacije u okviru takvog softverskog rešenja.

6. Metode zaštite

Potpuna zaštita od reverznog inženjeringa ne postoji. Onog trenutka kada se aplikacija nađe na memoriji uređaja, bez obzira na platformu, njene mašinske instrukcije su čitljive i uz manje ili više truda može se rekonstruisati izvorni kod. Nad Java virtuelnom mašinom je čitav proces još i jednostavniji, pa se zaštita koda svodi na njegovo maskiranje, ne bi li se otežao i usporio proces reverznog inženjeringa.

6.1. Maskiranje koda

Maskiranje koda je proces modifikovanja programskog koda u cilju smanjenja čitljivosti i razumevanja iz ugla programera, a bez gubitka funkcionalnosti na mašinskom nivou.
Neke od standardnih tehnika maskiranja su brisanje praznih karaktera (whitespace) u kodu, zamena imena promenljivih, šifrovanje stringova, samogenerišući kod ili čak ubacivanje programske logike koja ne služi ničemu, već samo dodatno zbunjuje programera u procesu tumačenja.
U zavisnosti od korišćenih tehnika, maskiranje može imati uticaj na veličinu i brzinu izvršavanja koda, ali takvi detalji neće biti opisivani u ovom radu. Fokus je stavljen na konkretne korake potrebne za zaštitu koda Android aplikacija, što će biti opisano u narednim sekcijama.

6.2. Detekcija modifikacije koda

Još jedan od klasičnih načina zaštite koda: generisanjem “hash” vrednosti aplikacionih fajlova i proverom sa pravom vrednosti pri startovanju aplikacije, može de detektovati promena u kodu i tako onemogućiti funkcionisanje aplikacije ili čak alarmirati autor.
Ova tehnika, naravno, takođe nije nezaobilazna, ali svakako poboljšava sigurnost i pravi dodatni problem napadačima pri kompromitovanju.

6.3. Odvajanje logike aplikacije na server

Ukoliko se aplikacija povezuje sa internetom, najbolji način čuvanja tajnosti algoritama je njihova selidba na server i njihovo korišćenje putem nekog servisnog interfejsa.
Na serveru se mogu proveravati i licencni podaci korisnika uređaja, što samo dodatno poboljšava kredibilnost identiteta.

6.4. Digitalno potpisivanje

Android platforma zahteva da sve instalirane aplikacije budu digitalno potpisane sertifikatom čiji privatni ključ poseduje samo vlasnik aplikacije. Android sistem koristi sertifikate za identifikovanje autora aplikacije i uspostavljanje poverenja između aplikacija. Digitalno potpisivanje ne može ni na koji način zaštiti aplikaciju od uvida u njen kod, ali pruža korisniku sigurnost da pokreće originalnu aplikaciju.
Potpisivanje privatnim ključem je integrisano u sva veća razvojna okruženja koja podržavaju Android i aktivira se pri kompajliranju u *release* modu.

6.5. Pregled konkretnih alata za zaštitu koda

6.5.1. ProGuard

ProGuard je verovatno jedan od najkompletnijih beplatnih alata za maskiranje Java koda. Osim što maskira kod, ProGuard ga i optimizuje, smanjuje i briše nepotrebne korake, rezultujući manjim APK fajlom značajno otežanim za razumevanje. ProGuard je već deo Android "build" sistema, čime je olakšana njegova upotreba, a pokreće se samo pri kompajliranju aplikacije u "release" modu. Korišćenje je opciono, ali je naravno jako preporučeno.

6.5.2. DexGuard

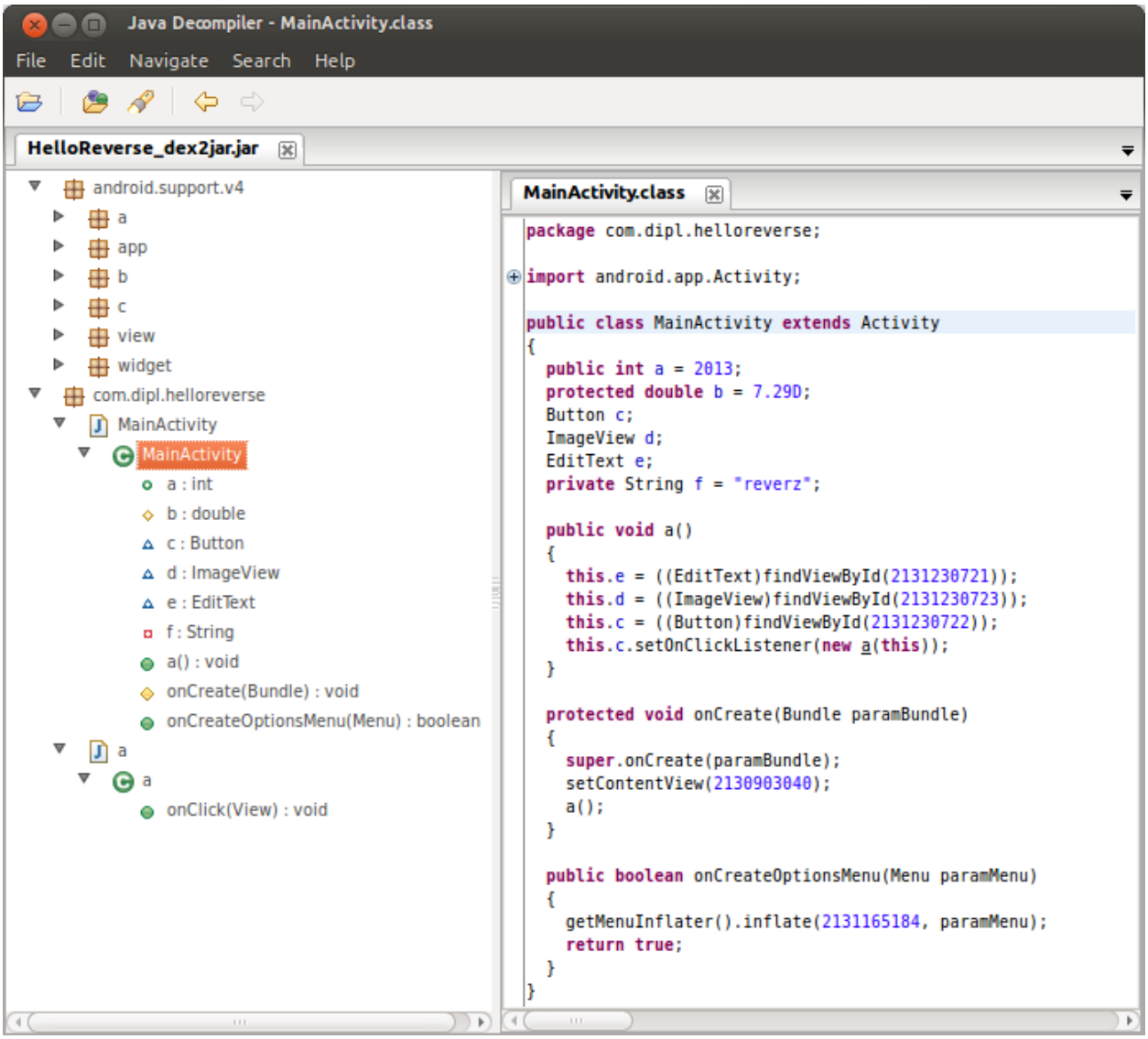
DexGuard je komercijalna alternativa ProGuard-u, napravljena pod krovom iste kompanije, koja pored osnovnih optimizacija i dubljeg maskiranja, dodatno šifruje stringove, klase, resurse i nudi mogućnnosti detekcije modifikacije koda od strane neovlašćenih lica.

6.5.3. Ostali alati

Postoji još mnoštvo alata za maskiranje Java koda koji su stekli dosta manje popularnosti u Android ekosistemu, a neki od njih su: yGuard, JODE, Java Guard, Allatori, DashO itd.

6.6. Primer dekompajliranja zaštićenog koda

Nad aplikacijom koja je napravljena za potrebe ovog rade (istom onom koja traži šifru i prikazuje sliku korisniku ako je ispravna) primenjeno je maskiranje koda iz ProGuard alata. Na slici se može videti rezultat maskiranja *MainActivity* klase, metoda, kao i drugačije imenovanje korišćenih paketa.



Slika 6.1. - JD-GUI prikaz dekompajliranog maskiranog koda.

Kao što se može primetiti, kod je svakako kompaktniji, znatno manje razumljiv, ali struktura i logika su ostale iste, a hardkodovane promenljive i dalje čitljive. Rezultat bi, bar po pitanju stringova, bio malo bolji korišćenjem DexGuard alata, ali kako je to komercijalni softver, nije bio korišćen u svrhu pisanja ovog rada.

7. Legalnost postupka i etika

Koncept reverznog inženjeringa izvan IT sveta ima dugu istoriju prihvaćene veštine. On predstavlja proces saznavanja i istraživanja svet oko nas, bez obzira na činjenicu da živimo u svetu koji je u značajnoj meri kreirao i čovek. Gledano iz tog ugla, svaki drugačiji stav može zvučati nelogično, ali u poslednjih nekoliko decenija, primene ovakvih tehnika se osporavaju, a sudski procesi uzrokovani njima prestali su da budu retkost.

7.1. Pravni aspekti reverznog inženjeringa

Univerzalno rešenje, naravno, ne postoji, i svaki pravni sistem definiše svoja pravila. Bitno je napomenuti da su turbulencije na ovom polju i dalje česte, što samo pokazuje da je sistem koji se primenjuje daleko od savršenog.

U Sjedinjenim Američkim Državama na primer, čak iako je proizvod zaštićen poslovnim tajnama, dozvoljeno je njegovo "rastavljanje na delove" dok god je proizvod nabavljen legalno. Sistem zaštite patenata i autorskih prava u SAD se umnogome i oslanja na tehnike reverznog inženjeringa, koristeći dobijene informacije kao dokaze u sudskim procesima. Sa druge strane, zanimljivo je primetiti da je takva praksa direktno kršenje EULA (End-user license agreement) sporazuma, koji često eksplicitno zabranjuju postupke za dublje razumevanje softverskih rešenja. U Pravnom sistemu SAD dakle, iako se isključuju - oba pravila važe, pa se tek sudskim postupcima utvrđuje ispravnost ovakvih metoda.

Evropska Unija je usvojila za nijansu jasnije zakone, koji dozvoljavaju reverzni inženjering za potrebe interoperabilnosti softvera, ali ga zabranjuju u svrhe stvaranja konkurentskog proizvoda. Takođe, zabranjena je bilo kakva javna objava informacija koje su dobijene ovakvim tehnikama.

7.2. Etičko pitanje

Ovaj problem je bio i ostaće predmet dugih debata, bez jasnih izgleda za konačan odgovor. Osnovni argument protiv primene reverznog inženjeringa je intelektualna svojina. Ako je individua ili organizacija stvorila proizvod ili ideju, da li je uredu da drugi "rasklapaju" taj proizvod u cilju otkrivanja njegovih tajni? Velike kompanije, koje ulažu znatno vreme i novac u razvoj svojih proizvoda, uglavnom ne misle tako. Zašto bi proizvođači ulagali svoje resurse u izgradnji intelektualne svojine, ako konkurenti reverznim inženjeringom mogu ukrasti njihove rezultate uz minimalni trud?
Sa druge strane, brojni su i benefiti ovakvih postupaka. Reverzni inženjering može biti korišćen u svrhe interoperabilnosti proizvoda, a možda i bitnije, u svrhu provere softvera na bilo kakve štetne, neetičke ili ilegalne aktivnosti, koje na drugi nači ne bi mogle biti otkrivene.

Globalni trend u svetu koji promoviše slobode, transparentnost i liberalizovano tržište, verovatno nam može predvideti i rešenje ovakvog pitanja. Do tada je tema otvorena za debatu.

8. Zaključak

Android platforma je svakako postala interesantno polje istraživanja i vredno tržište kompanijama. Kako aplikacije za ovu platformu postaju sve ozbiljnije, podatke je potrebno sve bolje čuvati, a kompanije polako ali sigurno počinju to i da prepoznaju.

Iskustvo je pokazalo da se od programera ne može uvek očekivati dovoljno poznavanje sigurnosnih metoda, pogotovo ako projektni menadžeri na njima ne insistiraju i ne rade ništa kako bi edukovali osoblje.

Svest se polako menja, pa se sve više razmišlja "unapred" i govori o sigurnosti kao temi koja se ne sme zanemariti. Poslodavcima je sada jasno da se sigurnosne metode moraju uključiti direktno u početne cikluse razvoja softvera, a da konstantni treninzi i edukacija postaju nužna praksa.

Savršenog rešenja nema, pošto je jasno da su napadači uvek spremni da odu korak dalje, ali se rizici mogu značajno smanjiti čak jednostavnim poznavanjem platforme i razumevanjem najčešćih napada.

8.1. Evaluacija rada i predlozi

Reverzni inženjering je samo jedna od metoda koje mogu uticati na sigurnost Android aplikacija i ovaj rad je stavio strogi fokus na nju. To ne sme zavarati čitaoca u pomisli da je to najrizičnije polje sigurnosnih propusta, niti najrasprostranjenije. Postoji još mnogo aspekata napada na Android platformu koji u ovde nisu spomenuti, i ovaj rad bi verovatno bio kompletniji da im je posvetio bar malo pažnje.

Treba primetiti i da reverzni inženjering spada pod tehnike statičke programske analize, dok postoje i drugi načini posmatranja ponašanja aplikacija, kroz dinamičke analize njihovih aktivnosti u vremenu izvršavanja. Istraživanje ovakve tematike bi bilo još kompleksnije i sigurno zahtevalo veće znanje, pa bi možda baš to bio logičan nastavak ovog rada u budućnosti.

9. Reference

1. Schulz, P. (2012), *Code Protection in Android*. PDF fajl. Bonn: University of Bonn.
2. Bridges, E., Kishore, D., Pedo, J. (2012). *Android App Security Analysis*. PDF fajl. New Jersey: Rutgers University, School Of Engineering.
3. Amalfitano, D. (2011). *Reverse Engineering and Testing of Rich Internet Applications*. PDF fajl. Naples: University of Naples Federico II.
4. @floyd_ch (2011). *Reversing Android Apps: Hacking and cracking Android apps is easy*. PDF fajl. Berne: Dreamlab Technologies.
5. Schulz, P., Matenaar, F. (2013). *Android Reverse Engineering & Defenses*. PDF fajl. San Francisco: Bluebox Security.
6. Huang, J. (2011). *Practice of Android Reverse Engineering*. PDF fajl. Taiwan: 0xlab.
7. Desnos, A., Gueguen, G. (2011). *Android: From Reversing to Decompilation*. PDF fajl. Laval: ESIEA - Operational Cryptology and Virology Laboratory.
8. Apvrille, A. (2012). *Android Reverse Engineering Tools, From an anti-virus analyst's perspective*. PDF fajl. Sunnyvale: Fortinet.
9. Díaz, V. A. (2013). *Android reverse engineering: understanding third-party applications*. PDF fajl. Bucharest: The OWASP Foundation.
10. Manjunath, V. (2011). *Reverse Engineering Of Malware On Android*. PDF fajl. Bethesda, MD: SANS Institute InfoSec Reading Room.
11. Joorabchi, M. E., Mesbah, A. (2012). *Reverse Engineering iOS Mobile Applications*. PDF fajl. Vancouver: University of British Columbia.
12. The Guardian. (2010). *A better way: explaining Android*. Veb. Jul 2013.
13. Teitelbaum, D. (2012). *Reverse Engineering Android: Disassembling Hello World*. Veb. Jul 2013.
14. Aher, S. (2013). *Patching/Modifying String within Native Android App*. Veb. Jul 2013.
15. Samuelson, P., Scotchmer, S. (2002). *The Law and Economics of Reverse Engineering*. PDF fajl. New Haven: The Yale Law Journal.
16. North Carolina State University. (bez datuma). *Reverse Engineering*. Veb. Avgust 2013.
17. Ostermeier, D., Sankey, J. (2010). *Understanding the Android Build Process*. Veb. Avgust 2013.