Primjena obrnutog inženjeringa u svrhu analize sigurnosti Android aplikacija

Ilić, Barbara

SVEUČILIŠTE U ZAGREBU
FAKULTET PROMETNIH ZNANOSTI

Barbara Ilić

PRIMJENA OBRNUTOG INŽENJERINGA U SVRHU ANALIZE SIGURNOSTI ANDROID APLIKACIJA

DIPLOMSKI RAD

Zagreb, 2020.

Sveučilište u Zagrebu

Fakultet prometnih znanosti

DIPLOMSKI RAD

PRIMJENA OBRNUTOG INŽENJERINGA U SVRHU ANALIZE SIGURNOSTI ANDORID APLIKACIJA

APPLICATION OF REVERSE ENGINEERING FOR THE PURPOSE OF ANALYZING THE SECURITY OF ANDROID APPLICATIONS

Mentor: dr. sc. Siniša Husnjak

Student: Barbara Ilić

Zagreb, 2020.

PRIMJENA OBRNUTOG INŽENJERINGA U SVRHU ANALIZE SIGURNOSTI ANDROID APLIKACIJA

SAŽETAK

Eksponencijalni rast i razvoj tehnologije te informatizacija i digitalizacija društva pridonijeli su porastu broja aktivnih korisnika mobilnih terminalnih uređaja, čime su mobilne aplikacije postale epicentar trenutnih razvojnih trendova. Cilj ovog diplomskog rada jest prikazati analizu sigurnosti Android aplikacija korištenjem postupka obrnutog inženjeringa. Postupkom obrnutog inženjeringa omogućuje se otkrivanje načina rada pojedine Android aplikacije, s ciljem čitanja i razumijevanja programskog koda te pronalaska potencijalnih sigurnosnih propusta i osjetljivih podatka unutar koda. Rad se temelji na prikazu tehnika i različitih alata te njihovih specifikacija namijenjenih provođenju obrnutog inženjeringa, kao i prikazu analize sigurnosti Android aplikacija na temelju praktičnog primjera.

KLJUČNE RIJEČI: obrnuti inženjering, sigurnost, zaštita, Android OS, mobilne aplikacije

APPLICATION OF REVERSE ENGINEERING FOR THE PURPOSE OF ANALYZING THE SECURITY OF ANDROID APPLICATIONS

SUMMARY

Exponential growth and development of technology and informatization and digitalization of society have contributed to the increase in the number of active users of mobile terminal devices, which has made mobile applications the epicenter of current development trends. The purpose of this paper is to present an analysis of the security of Android applications using the reverse engineering procedure. The reverse engineering procedure enables the detection of the mode of operation of an individual Android application, with the aim of reading and understanding the program code and finding potential security vulnerabilities and sensitive data within the code. The paper is based on the presentation of techniques and various tools and their specifications intended for the implementation of reverse engineering, as well as the presentation of security analysis of Android applications based on a practical example.

KEY WORDS: reverse engineering, security, protection, Android OS, mobile applications

Sadržaj

1. Uvod

Razvojem tehnologije, digitalizacijom i informatizacijom društva, mobilni terminalni uređaji te mobilne aplikacije bilježe porast broja aktivnih korisnika posljednjih nekoliko godina. Android kao OS otvorenog koda i dalje ima većinski udio na tržištu te većina krajnjih korisnika posjeduje mobilni terminalni uređaj zasnovan na upravo tom OS-u. S obzirom na to da je Android OS otvorenog koda gdje krajnji korisnik ima mogućnost ispravljanja pogrešaka u sustavu, prisutna je određena razina rizika. Android aplikacije razvijane su većinom u Java programskom jeziku, pri čemu je korištenjem različitih specijaliziranih alata moguće pristupiti izvornom kodu aplikacije. Taj postupak naziva se postupkom obrnutog inženjeringa (engl. *reverse engineering*), a provodi se u različite svrhe, pri čemu je jedna od njih analiza sigurnosti same aplikacije te pronalazak potencijalnih sigurnosnih propusta unutar koda.

Naslov diplomskog rada jest *Primjena obrnutog inženjeringa u svrhu analize sigurnosti Android aplikacija*, a sam rad strukturiran je kroz 8 poglavlja, uključujući uvod i zaključak:

1. Uvod
2. Mobilni operativni sustav Android
3. Komponente Android aplikacije i struktura APK datoteke
4. Obrnuti inženjering u svrhu sigurnosti Android aplikacija
5. Alati za provođenje statičke analize Java izvornog koda
6. Praktični primjer obrnutog inženjeringa Android aplikacije
7. Zaštita aplikacija od obrnutog inženjeringa
8. Zaključak

U drugom poglavlju rada objašnjene su i detaljno opisane ključne komponente od kojih se sastoji mobilni OS Android. Također, objašnjena je podjela mobilnih OS-a na OS otvorenog te zatvorenog koda, kao i podjela na temelju postojećih OS-a koje koriste različita računala. U drugom poglavlju rada također je prikazana arhitektura trenutno najkorištenijeg i vodećeg mobilnog OS Android prema slojevima te je objašnjen Android sigurnosni model, pri čemu se sigurnost Android OS u osnovi oslanja na sigurnosnu značajku omogućenu od strane Linux jezgre.

U trećem poglavlju rada objašnjen je i prikazan način rada mobilnih aplikacija te od kojih se komponenata sastoji svaka Android aplikacija. Android aplikacije većinom su razvijane u

Java programskom jeziku, te su pohranjene u .apk formatu, pri čemu je struktura APK datoteke prikazana i objašnjena u poglavlju 3.

U četvrtom poglavlju rada objašnjen je postupak provođenja obrnutog inženjeringa te su specificirani razlozi provođenja tog postupka. Također, prikazana je podjela na statičku i dinamičku analizu te je detaljno pojašnjena svaka od navedenih.

U petom poglavlju rada prikazane su osnovne značajke nekih od najpoznatijih alata za provođenje statičke analize izvornog koda Android aplikacija.

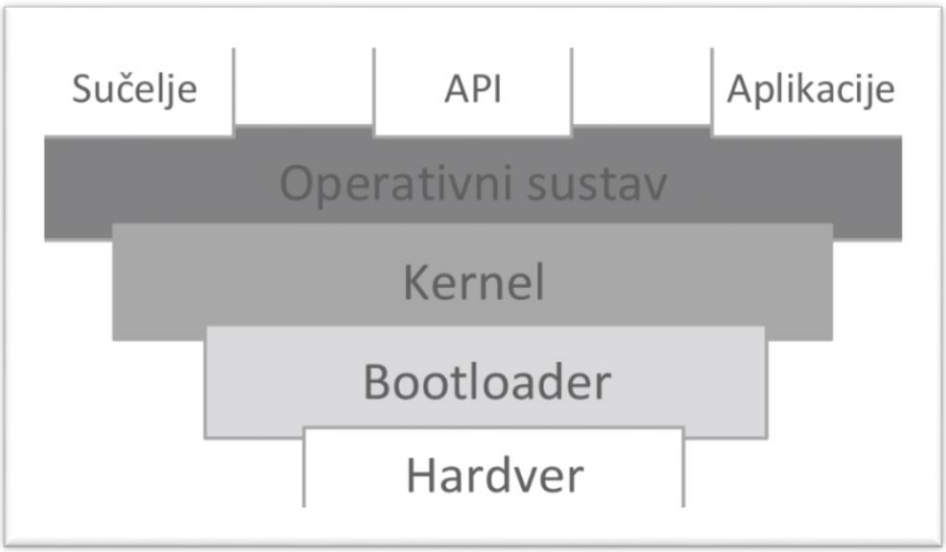
U šestom poglavlju rada prikazan je postupak obrnutog inženjeringa kroz praktičan primjer na dvije Android aplikacije: WaTF Bank, aplikacije koja simulira funkcionalnosti aplikacije mobilnog bankarstva te CovidLock, aplikacija namijenjena praćenju pojave korona virusa u stvarnom vremenu, a koja zapravo u sebi sadrži ucjenjivački softver.

Postupak obrnutog inženjeringa, osim u legitimne svrhe provjere sigurnosti, koristi se i u druge svrhe kao što su primjerice izmjena ili krađa koda aplikacije. Sedmo poglavlje rada prikazuje načine zaštite koda Android aplikacija od provođenja postupka obrnutog inženjeringa. Također, u sedmom poglavlju prikazane su značajke alata ProGuard namijenjenog zaštiti izvornog koda aplikacije.

Na kraju rada, u osmom poglavlju, donesen je jedinstven i subjektivan zaključak na temelju napisanih poglavlja te provedenog praktičnog dijela.

2. Mobilni operativni sustav Android

Razvojem informacijsko-komunikacijske tehnologije, mobilni terminalni uređaji postali su dio čovjekove svakodnevnice, pri čemu velika većina krajnjih korisnika u vlasništvu ima više od jednog pametnog mobilnog terminalnog uređaja trenutno. Industrija pametnih uređaja kontinuirano eksponencijalno raste, kako u veličini tržišta, tako i u modelima i dobavljačima. Shodno tome, pametni mobilni terminalni uređaj, u definiciji, predstavlja mobilni uređaj s naprednijim računalnim mogućnostima i mogućnostima povezivanja u odnosu na starije oblike mobilnih uređaja, [1]. Na slici 1 prikazana je struktura hardvera mobilnih terminalnih uređaja prema slojevima.



Slika 1. Struktura hardvera mobilnih terminalnih uređaja, [2]

Sa slike 1 moguće je uočiti kako se svaki mobilni terminalni uređaj sastoji od nekoliko ključnih komponenata: sučelja, aplikacijskog programskog sučelja (engl. *Application Programming Interface* – API), aplikacija, OS-a, jezgre (engl. *kernel*), tzv. *bootloader*-a te hardverskih komponenata. Aplikacijsko programsko sučelje predstavlja skup definicija i protokola namijenjenih izgradnji i integriranju aplikacijskog softvera te omogućuje komunikaciju između različitih proizvoda i usluga, [3]. *Bootloader,* nadalje, predstavlja program, odnosno kod, zadužen za pokretanje operativnog sustava uređaja te određuje redoslijed pokretanja operativnog sustava, [2].

Mobilni OS, u definiciji, predstavlja softver koji pokreće i upravlja mobilnim terminalnim uređajima, pri čemu ga je, prema [4], moguće podijeliti u dvije osnovne kategorije:

- OS otvorenog koda (engl. *open source*) koji omogućavaju instalaciju originalnog koda besplatno na uređaj, pri čemu sam krajnji korisnik ima mogućnost ispravljanja pogrešaka u sustavu, s ciljem poboljšanja performansi OS-a, a primjer OS-a otvorenog koda jest Android OS, te
- OS zatvorenog koda (engl. *closed source*), koji zabranjuju izmjene pri čemu isključivo originalni programer ima potpunu kontrolu te se za razliku od sustava otvorenog koda, ovaj sustav ne može fleksibilno mijenjati. Primjer OS-a zatvorenog koda jest iOS, razvijen od strane kompanije *Apple*.

OS najkritičniji je i ključni element bilo kojeg procesorski baziranog uređaja, a koristi se za upravljanje hardverskim i softverskim resursima unutar uređaja, kontrolu i izvođenje osnovnih zadataka te upravljanje različitim izvršavajućim procesima. Mobilni OS-i, prema [6], mogu se klasificirati na temelju postojećih operativnih sustava koje koriste različita računala, na:

- OS u stvarnom vremenu (engl. *Real-Time Operating System* – RTOS): reagiraju na unos u stvarnom vremenu istovremeno generirajući rezultate. RTOS sustavi obično se koriste u znanstvenim uređajima i sličnim malim instrumentima gdje su memorija i resursi ključne komponente, pri čemu ovi uređaji imaju vrlo ograničene ili nulte korisničke alate.
- Jedan korisnik, jedna operacija (engl. *Single User, Single Tasking Operation System* – SUST OS): omogućuju krajnjem korisniku upravljanje procesom istodobno, pri čemu je nedostatak SUST OS-a teško izvođenje više operacija istovremeno.
- Jedan korisnik, više operacija (engl. *Single User, Multi Tasking Operating System* – SUMT OS): omogućava krajnjem korisniku pokretanje više od jednog programa ili procesa istovremeno, te
- Višekorisnički OS-i (engl. *Multi-User Operating System*): omogućavaju istovremeni rad jednog ili više korisnika, pri čemu neki OSi podržavaju rad stotine istovremenih korisnika.

OS također je odgovoran za upravljanje i pravilno korištenje memorije te za komunikaciju unutar uređaja, pri čemu krajnji korisnik obično nema izravnu interakciju sa samim operativnim sustavom, [6].

Komponente svakog mobilnog OS-a, prema [7], čine četiri temeljna sloja:

1. Jezgra (engl. *Kernel*),

2. Središnji sloj (engl. *Middleware*),
3. Pokretačke aplikacije (engl. *Runtime Applications*), te
4. Korisničko sučelje (engl. *User Interface*).

Najniži apstrakcijski sloj OS-a čini jezgra, koja predstavlja sučelje između hardvera i softvera uređaja. Jezgra ima pristup svim resursima uređaja te je zadužena za kontrolu resursa, upravljanje procesima, memorijom i diskom te pružanje usluga vezanih za njih drugim aplikacijama. Bitna karakteristika jezgre jest mogućnost davanja direktnog pristupa aplikaciji, pri čemu postoji više različitih vrsta jezgri, kao što su primjerice monolitne, mikro, hibridne, nano jezgre i sl., [2].

Središnji sloj (engl. *Middleware*) predstavlja skup modula koji omogućavaju postojanje mobilnih aplikacija, potpuno je pregledan krajnjem korisniku te nudi ključne usluge kao što su mehanizmi za razmjenu poruka, tumačenje web stranica, upravljanje uređajem i sigurnost, [8].

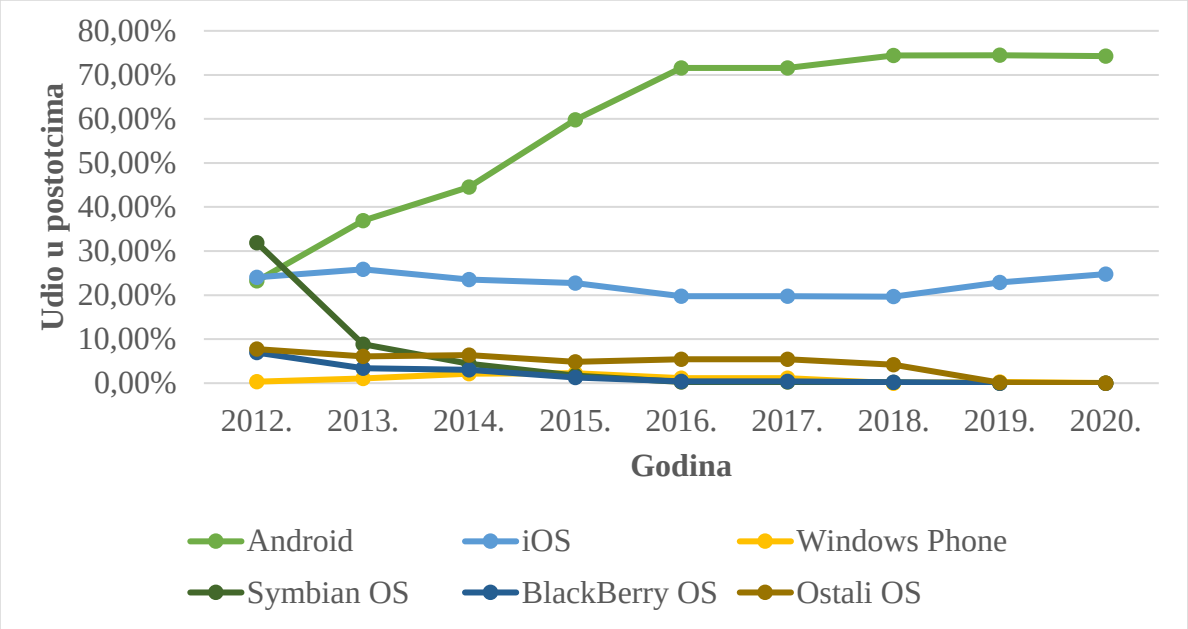
2.1. Razvoj Android operativnog sustava

Android predstavlja prvi OS otvorenog koda, temeljen na Linux platformi, razvijen od strane tvrtke *Google*, a namijenjen za primjenu na širokom spektru mobilnih terminalnih uređaja. Google-ova vizija jest da Android OS bude otvoren i besplatan te je iz tog razloga većina Android koda objavljena pod licencom *Apache*[1] otvorenog koda, što znači da svatko tko želi koristiti Android može preuzeti puni izvorni kod. Kao glavna prednost Android operativnog sustava ističe se jedinstveni pristup razvoju mobilnih aplikacija, [9].

Prema statistikama, Android OS činio je više od 80% svih prodaja pametnih mobilnih terminalnih uređaja širom svijeta 2017. godine te se procjenjuje kako će zadržati vodeće mjesto na tržištu, čineći oko 85% isporuka pametnih telefona 2020. godine, širom svijeta, [10].

[1] *Apache* licenca: licenca slobodnog i otvorenog koda (engl. *free and open source software*, FOSS) softverske fondacije *Apache Software Foundation* (ASF), kojom su propisani uvjeti upotrebe, reprodukcije, izmjena i distribucije bilo kojeg softvera izdanog pod licencom *Apache*

Na grafikonu 1 prikazan je kretanje tržišnog udjela najkorištenijih mobilnih OS-a u svijetu, u razdoblju od 2012. do siječnja 2020. godine.



Graf 1. Tržišni udio mobilnih OS-a u svijetu u razdoblju 2012.-2020.

Izvor: *[10], [3]*

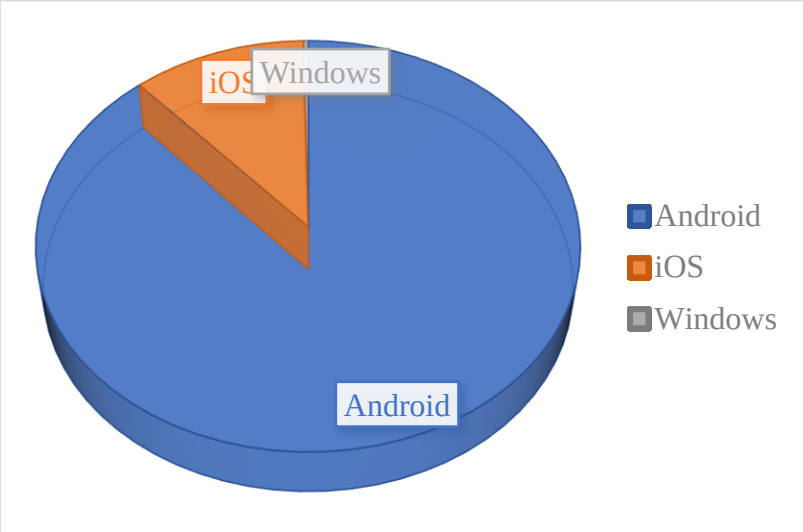
Sa grafikona 1 vidljivo je kako Android OS bilježi najveći tržišni rast u razdoblju od 2012. do 2020. godine. Analizirajući i uspoređujući podatke sa grafikona 1, vidljivo je kako ostali OS-i bilježe pad ili stagnaciju tržišnog udjela u promatranom razdoblju.

Budući da je Android OS otvorenog koda sa mogućnošću fleksibilnog prilagođavanja, ne postoji fiksna konfiguracija hardverskih i softverskih komponenti. Međutim, sam Android, prema [9], podržava slijedeće značajke:

- **Pohrana:** za pohranu podataka Android koristi SQLite relacijsku bazu podataka,
- **Povezivanje:** Android uređaji imaju razvijenu podršku za GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, *Bluetooth*, Wi-Fi, LTE i WiMax tehnologije,
- **Poruke:** podržava SMS i MMS poruke,
- **Web preglednik:** na temelju otvorenog koda WebKit, zajedno sa Chrome V8 JavaScript-om,
- **Podrška za medije:** uključuje podršku za H.263, H.264 (u 3GP ili MP4), MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF i BMP,

- **Hardverska podrška:** uključuje senzore akcelerometra, kameru, digitalni kompas, senzore blizine i GPS,
- **Dijeljenje:** podržava dijeljenje internetskih veza žično ili bežično.

Na grafikonu 2 prikazan je tržišni udio pojedinog OS-a na području Hrvatske, na temelju statističkih podataka iz ožujka 2020.



Graf 2. Tržišni udio OS-a u Hrvatskoj, ožujak 2020., *[11]*

Sa grafikona 2 vidljivo je kako najveći tržišni udio zauzima Android OS, te da ostali OS-i, osim *Apple*-ovog iOS-a te Windows OS-a, nisu zastupljeni u promatranom razdoblju i na promatranom području gotovo ni u kakvoj mjeri.

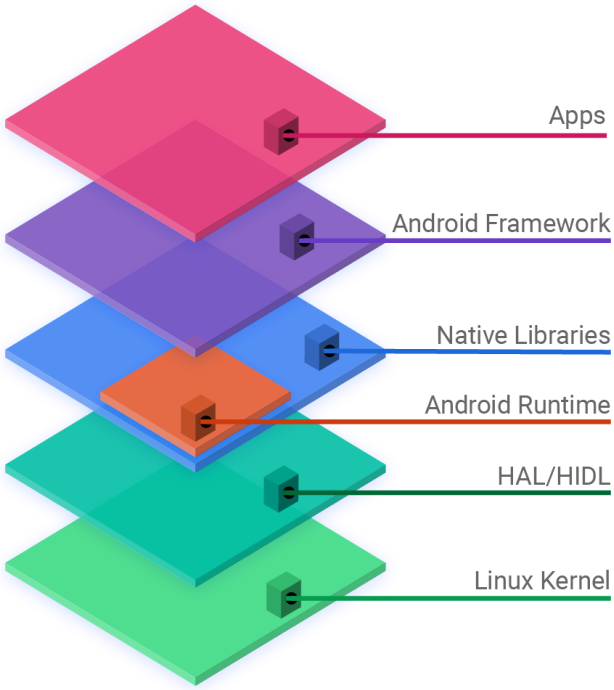
2.2. Arhitektura Android operativnog sustava

Arhitekturu Android OS-a čini softverski skup komponenata namijenjenih podršci širokom spektru mobilnih terminalnih uređaja. Središnji i ključni dio Android OS-a čini jezgra (engl. *kernel*) koja predstavlja sučelje između hardverskih i softverskih komponenata. Shodno tome, Android aplikacijski paket u osnovi je sastavljen od Linux jezgre, zbirke C/C++ knjižnica izloženih kroz okvir aplikacijskih usluga, ART virtualnog stroja (engl. *Android Runtime*) te sistemskih aplikacija, [5]. U konačnici, glavne komponente Android operativnog sustava, prema [5], uključuju sljedeće slojeve:

- Aplikacije,
- Android radni okvir (engl. *Android Framework*),
- ART virtualni stroj (engl. *Android Runtime*),
- Knjižnice platformi (engl. *Platform Libraries*),

- Sloj hardverske apstrakcije (engl. *Hardver Abstraction Level*) te
- Linux jezgru (engl. *Linux Kernel*).

Osnove komponente Android aplikacijskog paketa prikazane su slikom 2.



Slika 2. Komponente Android operativnog sustava

Izvor: *[12]*

Sa slike 2 vidljivo je kako Android OS čini više slojeva nužnih za omogućavanje temeljnih funkcionalnosti mobilnih terminalnih uređaja. Osnovu Android aplikacijskog paketa čini Linux jezgra koja omogućava razinu apstrakcije između hardvera uređaja te sadrži sve bitne hardverski upravljačke programe poput kamere, tipkovnice, zaslona i sl., [12]. Sloj hardverske apstrakcije (engl. *Hardware Abstraction Level* - HAL) omogućava standardna sučelja koja otkrivaju mogućnosti hardvera uređaja na višoj razini Java API okvira. HAL se, pri tome, sastoji od više modula knjižnice, od kojih svaki implementira sučelje za pripadajuću vrstu hardverske komponente, poput kamere ili *bluetooth* modula. Kada API okvir zahtjeva pristup hardveru uređaja, Android sustav učitava modul knjižnice za tu komponentu hardvera, [1].

Na vrhu Linux jezgre nalazi se skup knjižnica (engl. *Libraries*), koje uključuju otvoreni izvorni web preglednik *WebKit*, bibliotečni *libc*, SQLite bazu podataka koja predstavlja spremište za pohranu i razmjenu podataka o aplikacijama, knjižnice za reprodukciju i snimanje video i zvučnih zapisa, SSL knjižnice odgovorne za internetsku sigurnost i sl., [13].

Kategorija Android biblioteka obuhvaća one biblioteke temeljene na Java programskom jeziku, specifične za razvoj Android operativnog sustava. Neke od ključnih središnjih biblioteka dostupne Android razvojnim programerima, prema [13], su:

- **android.app**: omogućuje pristup modelu aplikacija i temelj je svih aplikacija Android operativnog sustava,
- **android.content**: koristi se za pristup podacima koje objavljuju dobavljači sadržaja i uključuje klase upravljanja bazama SQLite,
- **android.opengl**: Java sučelje za OpenGL ES 3D grafički prikaz API-ja,
- **android.os**: omogućuje aplikacijama pristup standardnim uslugama operativnog sustava, uključujući poruke, sistemske usluge i međuprocesnu komunikaciju,
- **android.text**: koristi se za prikazivanje i upravljanje tekstom na zaslonu uređaja,
- **android.view**: temeljni sastavni dijelovi korisničkog sučelja aplikacija,
- **android.widget**: kolekcija unaprijed ugrađenih komponenti korisničkog sučelja, poput gumba, naljepnica, upravitelja izgleda i sl., te
- **android.webkit**: skup klasa namijenjenih omogućavanju ugradnje mogućnosti pregledavanja web stranica u aplikacije.

Za uređaje koji koriste Android verzije 5.0 i novije svaka se aplikacija pokreće u vlastitom postupku i sa vlastitom instancom *Android Runtime*-a (ART). ART ima mogućnost pokretanja više virtualnih računala na uređajima s malo memorije, izvršavanjem DEX datoteke[2], [1]. Ključni dio ART-a čini Dalvik virtualni stroj (engl. *Dalvik Virutal Machine*) posebno dizajniran i optimiziran za Android OS. Dalvik VM koristi temeljne Linux funkcije poput upravljanja memorijom, što je svojstveno Java programskom jeziku. Dalvik VM omogućava da se svaka Android aplikacija pokrene u vlastitom procesu uz vlastiti primjerak virtualnog stroja Dalvik. ART također nudi skup osnovnih knjižnica koje omogućuju Android programerima pisanje Android aplikacija korištenjem standardnog Java programskog jezika, [13].

Android radni okvir (engl. *Application Framework*) pruža skup usluga više razine aplikacijama u obliku Java klasa, pri čemu je programerima omogućeno da te usluge koriste u svojim aplikacijama. Ključne usluge koje čine Android okvir, prema [13] su slijedeće:

- **Upravitelj aktivnosti:** upravlja svim aspektima životnog ciklusa aplikacija,

[2] DEX datoteka, tehničkog naziva *Dalvik Executable*: izvršna datoteka spremljena u formatu koji sadrži kompajlirani kod napisan za Android

- **Davatelji sadržaja:** omogućuju aplikacijama objavljivanje i dijeljenje podataka s drugim aplikacijama,
- **Upravitelj resursa:** omogućuje pristup ugrađenim izvorima koji nisu kodni, kao što su postavke boja i izgled korisničkog sučelja,
- **Upravitelj obavijesti:** omogućuje aplikacijama da prikazuju upozorenja i obavijesti krajnjim korisnicima, te
- **Sustav prikaza (engl. *View System*):** što obuhvaća proširivi skup prikaza koji se koriste za stvaranje korisničkih sučelja aplikacija.

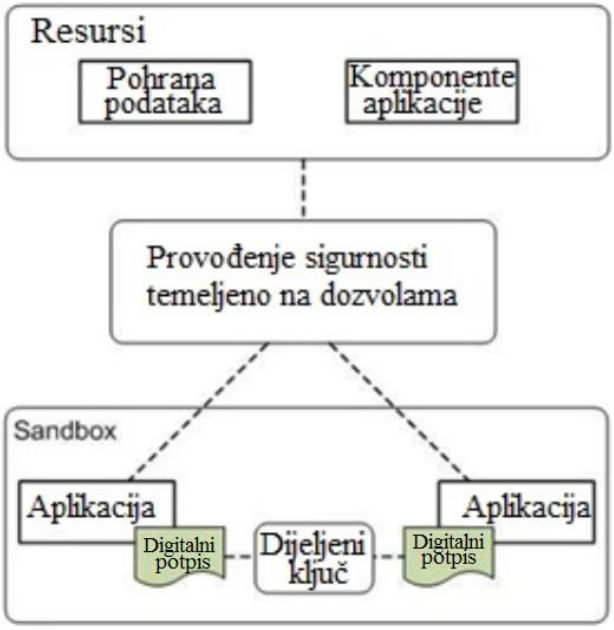
Svaki Android uređaj dolazi sa skupom osnovnih aplikacija za elektroničku poštu, SMS poruke, pregledavanje interneta, kontakte, pozive i slične usluge. Aplikacije uključene u platformu nemaju poseban status među aplikacijama koje krajnji korisnik odluči sam instalirati na uređaj. Shodno tome, aplikacije treće strane mogu postati korisnički zadani web preglednik, SMS centar ili zadana tipkovnica. Sistemske aplikacije, pri tome, funkcioniraju i kao korisničke aplikacije te pružaju ključne mogućnosti kojima programeri mogu pristupiti iz vlastite aplikacije, [1].

2.3. Android sigurnosni model (engl. *Android Security Model*)

Sigurnost Android OS-a u osnovi se oslanja na sigurnosnu značajku omogućenu od strane Linux jezgre; svaka Android aplikacija pokreće se kao zaseban proces u zasebnoj instanci Dalvik virtualnog stroja, s vlastitim skupom podataka, sprječavajući druge procese da ometaju njezino izvršavanje. Na aplikacijskoj razini, Android koristi preciznije dozvole kako bi se dopustila ili onemogućila interakcija aplikacija ili komponenata sa drugim aplikacijama i komponentama te kritičnim resursima, pri čemu je potrebna dozvola krajnjeg korisnika prije no što aplikacija dobije pristup kritičnim operacijama, kao što su ostvarivanje poziva ili slanje SMS poruka. Shodno tome svaka Android aplikacija zahtijeva dozvolu krajnjeg korisnika za pristup odgovarajućim resursima, nužnim za njezino uspješno izvršavanje, [31].

Android sigurnosni model (engl. *Android Security Model*) u osnovi, predstavlja izolirani model koji osigurava sigurnosno okruženje za izvođenje i pokretanje Android aplikacija. Android sigurnosni model primarno je baziran na tzv. *sandbox* mehanizmu, što podrazumijeva da se svaka aplikacija pokreće unutar vlastite instance Dalvik virtualnog stroja, pri čemu svaka aplikacija ima svoju jedinstvenu ID vrijednost, što osigurava izvođenje koda aplikacije izolirano i neovisno od kodova drugih aplikacija, [33]. Kako bi se osiguralo izolirano izvođenje i pokretanje aplikacija, Android koristi standardni Linux mehanizam kontrole pristupa, pri

čemu je svakom Android aplikacijskom paketu prilikom instalacije dodijeljen jedinstveni Linux korisnički ID, [34]. Shema Android sigurnosnog modela prikazana je slikom 3.

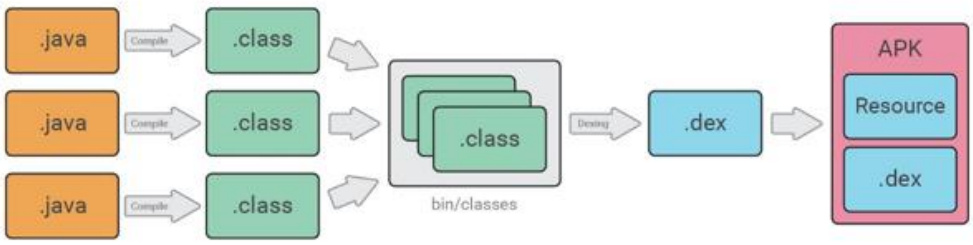


Slika 3. Android sigurnosni model, [32]

Sa slike 3 vidljivo je kako je svaka Android aplikacija potpisana digitalnim potpisom uz pomoć jedinstvenog privatnog ključa poznatog isključivo vlasniku, odnosno programeru same aplikacije. Nakon instalacije na uređaj, aplikaciji se dodjeljuje jedinstveni korisnički ID, čime se izbjegava utjecaj i modifikacija drugih aplikacija stvaranjem jedinstvenog *sandbox*-a za tu aplikaciju, pri čemu je dodijeljeni korisnički ID trajan. Na taj način osigurava se da zlonamjerna aplikacija ne može pristupiti, odnosno kompromitirati podatke originalne aplikacije. Predstavljeni sigurnosni model osigurava sigurnosno okruženje za izvođenje Android aplikacija, no ograničenja koja model primjenjuje smanjuju mogućnosti izvođenja aplikacije. Sigurnosni mehanizmi moraju omogućiti dovoljnu fleksibilnost programerima aplikacija, istovremeno osiguravajući dovoljnu sigurnost sustava. Kao što je vidljivo na slici 3, dvije aplikacije mogu dijeliti podatke i komponente aplikacije, primjerice, jedna aplikacija može pokrenuti aktivnost koja pripada drugoj aplikaciji ili pristupiti njezinim datotekama, [33].

3. Komponente Android aplikacije i struktura APK datoteke

Android aplikacije većinom su razvijane u Java programskom jeziku, a nakon što su jednom napisane aplikacije se kompajliraju (engl. *compiling*) u DEX kod (engl. *Dalvik EXecutable*). Aplikacije se potom izvršavaju kao posebni procesi u vlastitoj instanci Dalvik-a, što između ostalog omogućava neovisnost jedne aplikacije o drugoj odnosno, ukoliko se jedna aplikacija sruši, to neće utjecati na izvođenje na druge aplikacije. Dio Android sustava koji pokreće DEX kod jest Dalvik virtualni stroj, [14], [15]. Postupak kompajliranja Android aplikacije prikazan je slikom 4.



Slika 4. Pretvorba Java koda u .dex datoteku, [16]

Sa slike 4 vidljivo je kako postupak kompajliranja podrazumijeva dva ključna koraka:

1. pretvorbu Java koda u .class datoteke, te potom
2. pretvaranje .class datoteka u .dex format korištenjem dx alata.

Kod Android aplikacije, zajedno s bilo kojim podacima i datotečnim resursima sastavlja se u APK datoteku, koja predstavlja arhiviranu datoteku Android aplikacije s .apk sufiksom. APK datoteka sadrži sav sadržaj Android aplikacije te predstavlja datoteku koju uređaji bazirani na Android operativnom sustavu koriste prilikom instaliranja aplikacije, [17].

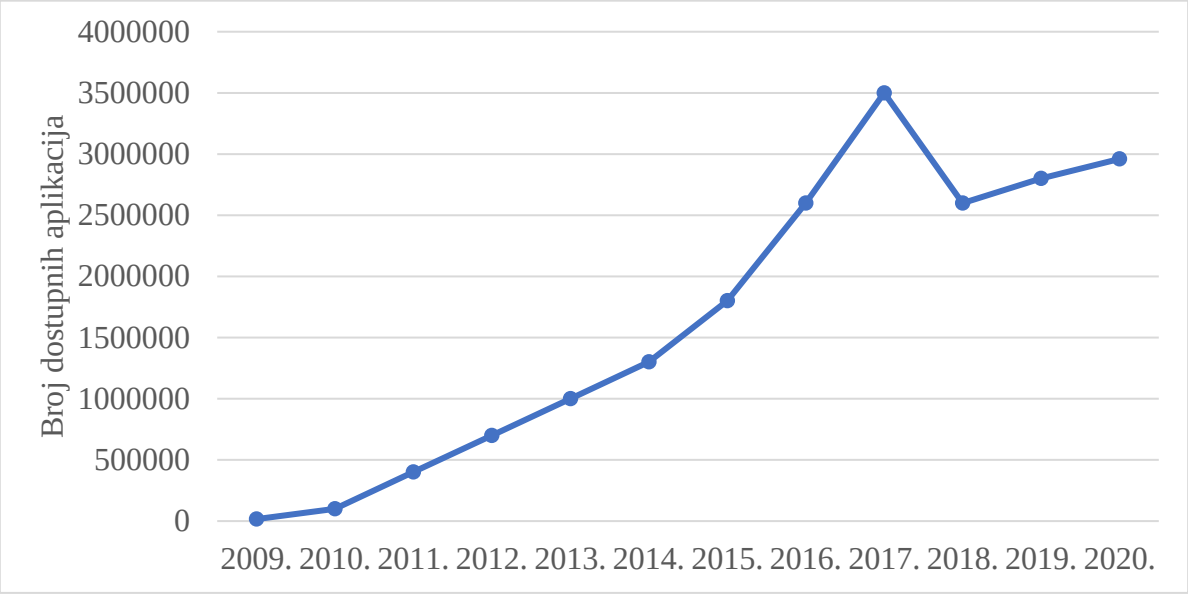
Svaka Android aplikacija izolirana je u svom sigurnosnom okruženju te je zaštićena slijedećim sigurnosnim značajkama:

- Android OS višekorisnički je OS baziran na Linux platformi, pri čemu svaka aplikacija predstavlja drugog korisnika.
- Sustav, po zadanom, dodjeljuje svakoj aplikaciji jedinstveni Linux korisnički identifikacijski broj (ID), koji je korišten isključivo od strane sustava te je nepoznat aplikaciji. Sustav određuje dopuštenja za sve datoteke u aplikaciji, na način da im može pristupiti isključivo korisnički ID dodijeljen toj aplikaciji.

- Svaki proces ima svoj zasebni virtualni stroj, tako da se kod svake aplikacije izvodi izolirano od drugih aplikacija.
- Prema zadanom, svaka aplikacija pokreće vlastiti Linux proces: Android OS pokreće proces kada bilo koja komponenta aplikacije treba biti izvršena te ga zatvara kada više nije potrebno izvršavanje ili u slučaju da sustav mora oporaviti memoriju za druge aplikacije.

Android OS radi na načelu najmanje privilegije, što znači da svaka aplikacija, prema zadanim postavkama, ima pristup samo onim komponentama koje su joj potrebne za njezino izvršavanje. Na taj način stvara se sigurno okruženje u kojem aplikacija nije u mogućnosti pristupiti dijelovima sustava za koje nema dopuštenje, [17].

Razvojem tehnologije, digitalizacijom i informatizacijom društva, mobilne aplikacije bilježe porast broja aktivnih korisnika posljednjih nekoliko godina. Shodno tome, na grafikonu 3 prikazan je broj aplikacija dostupnih unutar Google Play trgovine u razdoblju od 2009. do početka 2020. godine.

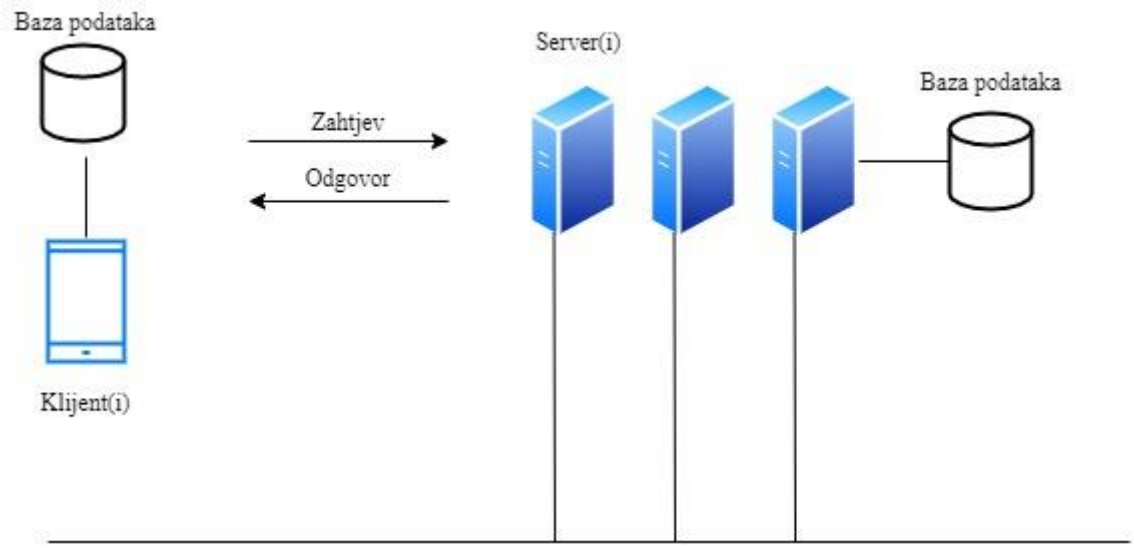


Graf 3. Broj dostupnih aplikacija unutar Google Play 2009.-2020., *[49]*

Na grafikonu 3 prikazana je statistika broja dostupnih aplikacija unutar trgovine Google Play u razdoblju od 2009. do 2020. godine. Sa grafikona 3 vidljivo je kako mobilne aplikacije bilježe rast u promatranom razdoblju, a broj dostupnih aplikacija nedavno je dosegnuo broj od preko 2 i pol milijuna, nakon što je u srpnju 2013. godine premašio broj od milijun dostupnih aplikacija.

3.1. Način rada mobilnih aplikacija

Mobilne aplikacije epicentar su trenutnih razvojnih trendova, pri čemu se većina mobilnih aplikacija temelji na klijentsko-poslužiteljskoj arhitekturi koja podrazumijeva veći broj klijenata, udaljenih procesora, koji traže i dobivaju uslugu izravno s centraliziranog poslužitelja. Na slici 5 prikazana je klijentsko-poslužiteljska arhitektura.



Slika 5. Klijentsko-poslužiteljska arhitektura, *[51]*

Sa slike 5 vidljivo je kako se klijentsko-poslužiteljska arhitektura sastoji od skupa usluga pruženih od strane poslužitelja te skupa klijenata koji te usluge koriste. Pri tome, klijenti moraju biti svjesni poslužitelja koji su im dostupni, ali uobičajeno im nije poznato postojanje drugih klijenata. Arhitektura mobilnih aplikacija najčešće se modelira u smislu klijentsko-poslužiteljske arhitekture, pri čemu jedan ili više klijentskih uređaja zahtijevaju informacije s poslužiteljskog uređaja te poslužitelj odgovara traženim informacijama, što je vidljivo na slici 5, [51]. Općenito govoreći, klijentsko-poslužiteljska arhitektura mobilnih aplikacija sastoji se od platforme za distribuciju aplikacija, mobilnog terminalnog uređaja, kanala za prijenos podataka te poslužitelja. Gledajući sa strane krajnjeg korisnika, klijenta instaliranog na pametnom mobilnom terminalnom uređaju predstavlja sama mobilna aplikacija sa kojom je krajnji korisnik izravno u interakciji. Sa druge strane, poslužitelja kao drugu komponentu klijentsko-poslužiteljske arhitekture razvija razvojni programer, pri čemu često tu ulogu obavlja isti softver koji je odgovoran za generiranje i obradu sadržaja na web mjestu. Drugim riječima, komponenta na strani poslužitelja najčešće je web aplikacija koja putem Interneta komunicira s mobilnim klijentom putem posebnog aplikacijskog programskog sučelja (API – engl. *Application Programming Interface*). Svrha poslužitelja jest pohrana i obrada informacija, kao

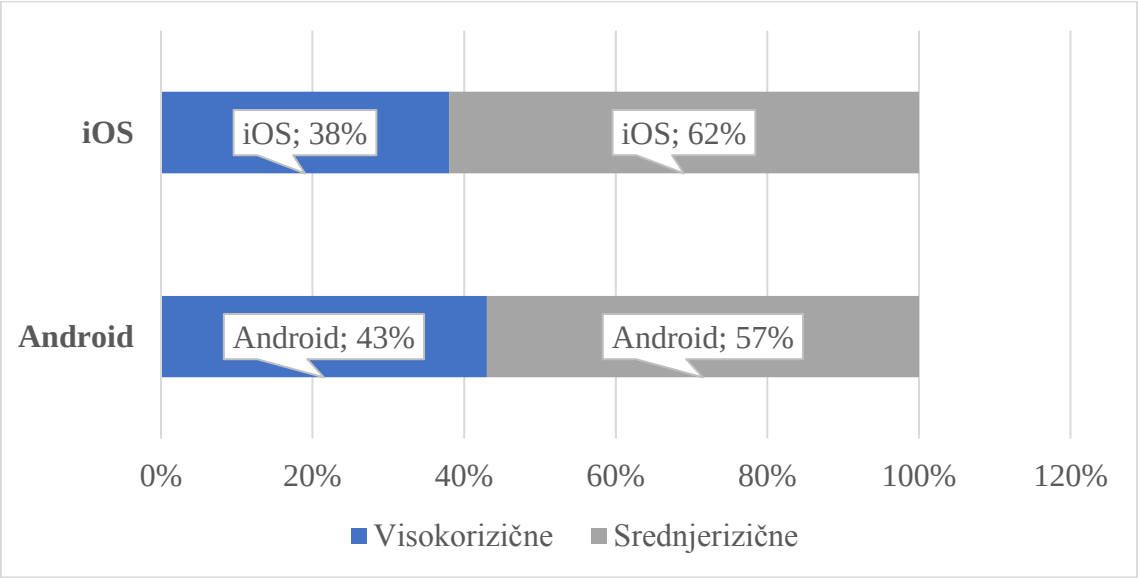
i sinkronizacija korisničkih podataka između uređaja. Noviji mobilni OS-i obuhvaćaju različite sigurnosne mehanizme, pri čemu, prema zadanim postavkama, instalirana aplikacija može pristupiti samo datotekama u vlastitim direktorijima tzv. sandbox-a, a korisnička prava ne dopuštaju uređivanje sistemskih datoteka. Ipak, pogreške koje su napravili razvojni programeri prilikom pisanja koda aplikacije uzrokuju nedostatke u zaštiti, pri čemu ih napadači mogu upotrijebiti u različite svrhe, [50].

Mobilne aplikacije uobičajeno je moguće podijeliti u dvije kategorije: unaprijed instalirane te korisnički instalirane aplikacije. Unaprijed instalirane aplikacije uključuju aplikacije ugrađene u terminalni uređaj od strane izvornog proizvođača opreme (engl. Original Equipment Manufacturer – OEM), kao što su kalendar, upravitelji elektroničke pošte, preglednik i kontakti. S druge strane, korisnički instalirane aplikacije predstavljaju aplikacije koje je krajnji korisnik sam preuzeo i instalirao, bilo putem trgovine Google Play ili ručno korištenjem ADB instalacije (engl. *Android Debug Bridge*). Android OS pri tome koristi poseban ključ u svrhu potpisivanja unaprijed instaliranih paketa aplikacija. Zatim se aplikacije treće strane potpisuju s ključevima koje je stvorio pojedinačni programer, pri čemu i za unaprijed instalirane i za korisnički instalirane aplikacije, Android OS koristi potpis za sprječavanje neovlaštenih ažuriranja aplikacija, [52].

3.2. Sigurnosne prijetnje usmjerene Android aplikacijama

Sigurnost Android OS-a temelji se na mehanizmu baziranom na dozvolama, koji regulira pristup Android aplikacijama i kritičnim resursima Android OS-a. Nedostatak mehanizma temeljenog na dozvolama jest gruba kontrola dozvola aplikacija, pri čemu primjerice krajnji korisnik mora prihvatiti sve zahtjeve za dozvole kako bi aplikaciju mogao instalirati na svoj terminalni uređaj, [53].

Budući da su Android mobilne aplikacije zasnovane na klijentsko-poslužiteljskoj arhitekturi, ranjivosti Android mobilnih aplikacija mogu se razvrstati na ranjivosti na strani klijenta te na strani poslužitelja. Pri tome se 60% ranjivosti nalazi sa klijentske strane, od čega 89% može biti iskorišteno bez ostvarivanja fizičkog pristupa, a 56% može biti iskorišteno bez ostvarivanja administrativnih prava (root-ani uređaji), [50]. Na grafikonu 4 prikazana je maksimalna razina rizika od ranjivosti na klijentskoj strani za operativne sustave Android i iOS.



Graf 4. Maksimalna razina rizika od ranjivosti na klijentskoj strani za OS Android i iOS, *[50]*

Android aplikacije uobičajeno sadrže kritične ranjivosti češće od aplikacija pisanih za iOS OS, što je vidljivo sa grafikona 4, ali ta razlika nije značajna te je ukupna razina sigurnosti klijenta mobilnih aplikacija za operativne sustave Android i iOS otprilike jednaka.

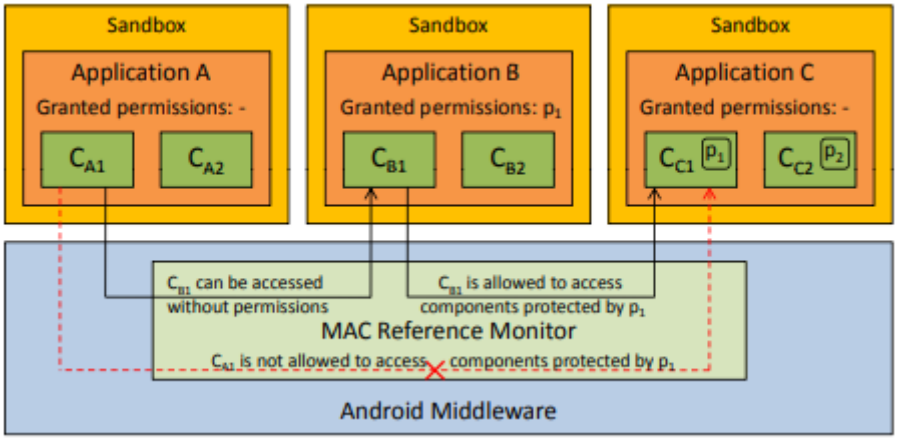
3.2.1. Curenje informacija

Prilikom instalacije Android aplikacija na mobilni terminalni uređaj, krajnji korisnik mora odobriti zahtjeve za pristup traženim resursima. Do curenja informacija dolazi kada krajnji korisnici daju dozvole za pristup resursima, bez ikakvih ograničenja od strane OS-a.

Mehanizam kontrole dozvola pokazao se neučinkovitim za zaštitu privatnosti krajnjih korisnika, budući da su istraživanja pokazala kako 70% aplikacija pametnih mobilnih terminalnih uređaja prikuplja podatke koji nisu ključni za osnovnu funkciju same aplikacije. Prilikom instalacije nove aplikacije mali broj korisnika obraća pažnju na tražene dozvole te jesu li one u skladu s funkcijom aplikacije, čime potencijalno mogu ugroziti svoju privatnost, što na kraju dovodi do curenja informacija, [53].

3.2.2. Eskalacija privilegija (engl. *Privilege escalation*)

Eskalacija privilegija predstavlja napad u vidu korištenja javno dostupnih ranjivosti jezgre Android OS-a, pri čemu aplikacija s manje dozvola (povlašteni pozivatelj) nije ograničena za pristup komponentama povlaštenije aplikacije, [54]. Slikom 6 prikazana je shema napada eskalacijom privilegija.



Slika 6. Napad eskalacijom privilegija, [54]

Pretpostavlja se kako se aplikacije A, B i C izvode na terminalnom uređaju zasnovanom na Android OS-u, a svaka od njih izolirana je unutar svog *sandbox-a*. Aplikacija A nema odobrenih dozvola te se sastoji od komponenata C_{A1} i C_{A2}, aplikacija B ima odobrenu dozvolu *p1* te se sastoji od komponenata C_{B1} i C_{B2}, pri čemu ni komponenta C_{B1} ni C_{B2} nisu zaštićene pomoću dozvola te tim komponentama može pristupiti bilo koja aplikacija. Aplikacija C nema odobrenih dozvola te se sastoji od komponenata C_{C1} i C_{C2}, pri čemu su komponente C_{C1} i C_{C2} zaštićene dozvolama *p1*, odnosno *p2*, što znači da komponenti C_{C1} može pristupiti isključivo aplikacija s dozvolom *p1*, dok komponenti C_{C2} može pristupiti isključivo aplikacija s dozvolom *p2*. Kao što je vidljivo na slici 6, komponenta C_{A1} ne može pristupiti C_{C1} komponenti izravno, budući da aplikacija A nema *p1* dozvolu. Međutim, podaci iz komponente C_{A1} mogu doseći komponentu C_{C1} neizravno putem C_{B1} komponente. Kako bi se spriječio ovakav oblik napada,

aplikacija B mora provesti dodatne provjere dozvola kako bi se osiguralo da aplikacija koja poziva komponentu C_{B1} daje dozvolu *p1*, [54].

3.2.3. Prepakiranje aplikacija (engl. *Repackaging Applications*)

Prepakiranje aplikacija (engl. Repackaging Applications) jedno je od najčešćih sigurnosnih pitanja Android OS-a. Prepakiranje predstavlja proces rastavljanja, odnosno raspakiravanja APK aplikacijskog paketa Android aplikacije korištenjem postupka obrnutog inženjeringa te tehnika umetanja zlonamjernog koda u izvorni kod aplikacije. Pri tome, tehnike prepakiranja aplikacija omogućavaju da se zlonamjerni kod prikriva kao izvorni kod aplikacije. Shodno tome, teško je razlikovati izvorni od zlonamjernog koda budući da prepakirana aplikacija obično funkcionira na isti način kao i legitimna. Koraci prepakiranja aplikacija, prema [53], su sljedeći:

- Otpakiranje (engl. *unpacking*): otpakiranje APK paketa korištenjem dostupnih alata na kojima se temelji postupak obrnutog inženjeringa,
- Dekompajliranje (engl. *decompiling*): što obuhvaća dekompajliranje Java izvornog koda pomoću JAD alata te ekstrakciju izvornog koda Java klase,
- Ubrizgavanje koda: ubrizgavanje koda i dodavanje resursa u glavni izvorni kod pomoću Java razvojnog okruženja, te
- Ponovno pakiranje: obnova datoteka pomoću *apktoola* i potpisivanje generiranih datoteka.

3.3. Komponente Android aplikacija

Komponente Android aplikacije bitni su sastavni dijelovi svake aplikacije, pri čemu svaka komponenta predstavlja ulaznu točku kroz koju sustav ili krajnji korisnik može pristupiti aplikaciji, a neke komponente ovise jedne o drugoj. Na slici 7 prikazane su osnovne komponente koje čine svaku Android aplikaciju.

Slika 7. Komponente Android aplikacije, *[18]*

Kao što je vidljivo na slici 7, razlikuju se sedam osnovnih komponenta svake Android aplikacije:

- Aktivnosti,
- Usluge,
- Dozvole,
- Fragmenti,
- Planovi (engl. *Intents*),
- Prijemnici odašiljanja (engl. *Broadcast recievers*) te
- Davatelji sadržaja i usluga.

Svaka od navedenih komponenata aplikacije ima svoju zadaću te jasan životni ciklus koji definira kako se komponenta stvara i uništava. Aktivnost, kao ključna komponenta Android aplikacije, označava ulaznu točku interakcije s krajnjim korisnikom te predstavlja zaslon prema korisničkom sučelju, odnosno vidljivi dio svake Android aplikacije. Iako aktivnosti djeluju zajedno kako bi stvorile kohezivno korisničko iskustvo unutar aplikacije, svaka aktivnost

neovisna je od preostalih. Aktivnost kao komponenta Android aplikacije olakšava ključne interakcije između sustava i korisnika:

- Praćenje onoga što trenutno korisnik koristi, kako bi se osiguralo da sustav nastavlja pokretati proces koji je domaćin te aktivnosti.
- Krajnji korisnik ima mogućnost vraćanja na prethodno pokrenute procese (zaustavljene aktivnosti) te se na taj način daje prioritet zadržavanju tih procesa.
- Pružanje načina da aplikacije provode korisničke protoke međusobno te da sustav koordinira te tokove, [17].

Komponenta aktivnosti definira se klasom *Activity*, pri čemu svaka aktivnost mora biti deklarirana unutar *AndroidManifest.xml* datoteke, korištenjem sintakse prikazane slikom 8.

```
<manifest ... >
  <application ... >
      <activity android:name=".ExampleActivity" />
      ...
  </application ... >
  ...
</manifest >
```

Slika 8. Sintaksa komponente aktivnosti unutar *AndroidManifest.xml* datoteke, [17]

Kao što je vidljivo sa slike 8, aktivnost se unutar *AndoridManifest.xml* datoteke deklarira kao dijete (engl. *child*) elementa <*application*>, pri čemu je jedini atribut koji je potrebno predati ovoj komponenti naziv, koji određuje naziv klase aktivnosti. Pri tome je moguće dodati i atribute koji definiraju karakteristike aktivnosti kao što su primjerice oznake, ikone ili teme korisničkog sučelja, [41].

Fragmenti predstavljaju posebno definirane dijelove aktivnosti koji sadrže dio korisničkog sučelja, međutim mogu biti korišteni isključivo i u svrhu upravljanja aktivnostima u pozadini sustava. Cilj fragmenata jest omogućiti ponovnu upotrebnu komponenata te adaptaciju različitim veličinama ekrana. Pri tome je moguće kombinirati više fragmenata unutar jedne aktivnosti kako bi se izradilo korisničko sučelje s više okna te kako bi se fragment ponovno iskoristio unutar više aktivnosti. Fragment pri tome uvijek mora biti smješten unutar komponente aktivnosti, a na životni ciklus fragmenta izravno utječe životni ciklus aktivnosti domaćina: primjerice, kada je aktivnost pauzirana, zaustavljeni su i svi fragmenti unutar nje, odnosno kada je aktivnost uništena, uništeni su i svi fragmenti smješteni unutar nje. Android je fragmente predstavio u 3.0 verziji OS-a (razina API-ja 11), prije svega radi podrške

dinamičnijem i fleksibilnijem dizajnu sučelja na velikim ekranima, poput tableta. Budući da je ekran tableta mnogo veći od onog na mobilnom terminalnom uređaju, postoji više prostora za kombiniranje i razmjenu korisničkog sučelja, a fragmenti omogućuju takav dizajn bez potrebe za upravljanjem složenijim promjenama hijerarhije prikaza, [43].

Usluga predstavlja ulaznu točku namijenjenu održavanju rada aplikacija, pokreće se u pozadini sustava i ne pruža korisničko sučelje, a omogućava izvršavanje dugotrajnih operacija ili obavljanje poslova za udaljene procese. Zahvaljujući usluzi moguće je, primjerice, reproducirati glazbu u pozadini, dok krajnji korisnik koristi neku drugu aplikaciju te je moguće dohvaćati podatke putem mreže, bez prekida interakcije korisnika s nekom aktivnošću. Komponenta usluga implementira se kao podklasa klase usluge kao što je prikazano na slici 9.

```
public class MyService extends Service {
}
```

Slika 9. Kod komponente usluga, [19]

Usluge je moguće klasificirati u dvije različite skupine: pokrenute usluge (engl. *started services*) te povezane usluge (engl. *bound services*). Pokrenute usluge zahtijevaju pokretanje od strane sustava dok njihova zadaća ne bude izvršena. Primjer pokrenute usluge jest sinkronizacija podataka u pozadini ili reprodukcija glazbe u pozadini ukoliko krajnji korisnik izađe iz aplikacije te istovremeno koristi neku drugu. S druge strane, povezane usluge pokreću se jer neka druga aplikacija ili sustav žele koristiti uslugu. U osnovi, povezana usluga predstavlja uslugu koja pruža aplikacijsko programsko sučelje (engl. *Application Programming Interface* – API) drugom procesu, pri čemu sustav zna da postoji ovisnost između tih procesa: ukoliko je proces *A* vezan za uslugu u procesu *B*, sustav zna da treba održati proces *B* i njegovu uslugu kako bi omogućio rad procesu *A* i obrnuto. Zahvaljujući fleksibilnosti, usluge predstavljaju koristan i ključan dio svake Android aplikacije, [17].

Dozvole omogućavaju aplikacijama pristup određenim komponentama uređaja, pri čemu je glavna svrha dozvole zaštita privatnosti Android korisnika. Shodno tome, svaka Android aplikacija mora zatražiti dozvolu za pristup osjetljivim korisničkim podacima, poput kontakata ili SMS poruka, kao i određenim značajkama sustava, poput kamere i interneta. Ovisno o značajki, sustav može automatski dati određenu dozvolu ili će tražiti krajnjeg korisnika odobravanje zahtjeva za pristup određenim značajkama. Pri tome je bitna karakteristika sigurnosne arhitektura Android operativnog sustava da niti jedna aplikacija, prema zadanim

postavkama, nema dozvolu za obavljanje bilo kakvih operacija koje bi štetno utjecale na druge aplikacije, OS ili samog krajnjeg korisnika, [42].

Svaka dozvola koju aplikacija zahtijeva mora biti deklarirana unutar *AndroidManifest.xml* datoteke, korištenjem oznake <*uses-permission*>, pri čemu je primjer sintakse dozvole prikazan slikom 10.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.example.snazzyapp">

    <uses-permission android:name="android.permission.SEND_SMS"/>

    <application ...>
        ...
    </application>
</manifest>
```

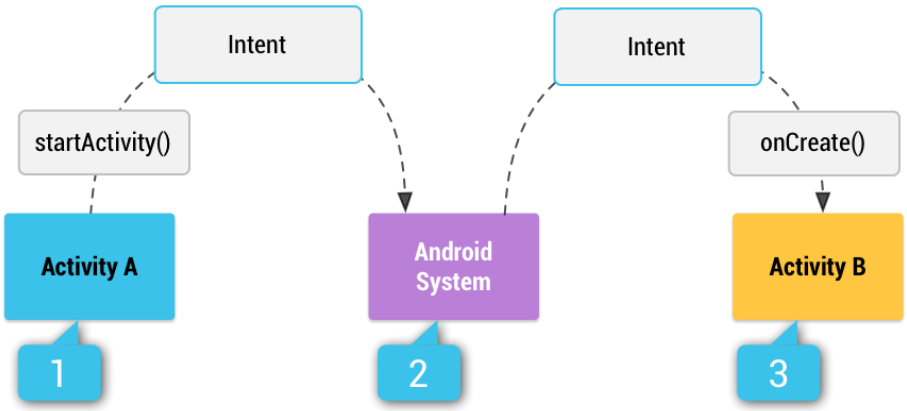
Slika 10. Primjer sintakse za komponentu dozvole, [42]

Slikom 10 prikazan je dio koda datoteke *AndroidManifest.xml* u kojem je deklarirana dozvola koja od krajnjeg korisnika zahtijeva pristup, odnosno slanje SMS poruka.

Komponenta prijamnici odašiljanja proširuje osnovnu klasu *BroadcastReciever* te ima samo jednu funkciju: slušati i odgovarati na najave odašiljanja. Aplikacije sustava Android u mogućnosti su pokrenuti emitiranje, no često emitiranje uključuje aktivnosti poput promjene vremenske zone ili upozorenja o niskoj razini baterije, koje nastaju na razini sustava. Aplikacija, pri tome, može imati bilo koji broj prijamnika odašiljanja, pri čemu emiter prijamnika ne prikazuje eksplicitno korisničko sučelje, ali obično generira vizualni znak za krajnjeg korisnika pokretanjem aktivnosti, [20].

Prema zadanim postavkama, podaci o aplikaciji u sustavu Android privatni su i nisu vidljivi drugim aplikacijama u sustavu, no aplikacija je u mogućnosti koristiti komponentu davatelji sadržaja, koja proširuje klasu *ContentProvider* te čini određeni dio podataka dostupnima drugim aplikacijama. Pri tome, aplikacije koje žele primiti podatke od *ContentProvider*-a ne pozivaju njegove metode izravno, već stvarajući objekt *ContentResolver* te koristeći njegove metode za dohvaćanje podataka od davatelja sadržaja. Objekt *ContentResolver* ima mogućnost suradnje s bilo kojim davateljem sadržaja što omogućuje upravljanje bilo kojim unutarnje procesnim komunikacijama koje su potrebne, [20].

Planovi (engl. *intents*) predstavljaju asinkronu komunikaciju kojoj je cilj omogućiti jednom objektu pozivanje radnji nekog drugog objekta. U datoteku *AndroidManifest.xml* unose se tzv. *intent* filteri kako bi se definirao tip planova kojeg komponente žele prihvaćati. Pri tome, razlikuju se dvije vrste planova: eksplicitni, koji daju ime komponenti koju žele pokrenuti te implicitni, koji se šalju operativnom sustavu kako bi se izvršila određena radnja nad skupinom podataka, [44]. Na slici 11 prikazano je kako se planovi koriste prilikom pokretanja aktivnosti.



Slika 11. Korištenje planova (engl. intents) prilikom pokretanja aktivnosti, [44]

Sa slike 11 vidljivo je kako implicitni plan putem sustava pokreće drugu aktivnost: Aktivnost A stvara namjeru s opisom akcije koju želi poduzeti te ju prosljeđuje u *startActivity()*, nakon čega Android sustav pretražuje sve aplikacije koje sadrže definirani filter namjere. Kada naiđe na podudaranje, sustav pokreće Aktivnost B pozivanjem metode *onCreate()* te prenosi željenu namjeru, [42].

3.4. Struktura APK datoteke

Svaka Android aplikacija pohranjena je u apk formatu, što u osnovi predstavlja arhiviranu zip datoteku, pri čemu je moguće promijeniti ekstenziju datoteke iz .apk u .zip te je raspakirati i vidjeti njezin sadržaj. Osnovne komponente APK datoteke prikazane su slikom 12.

Slika 12. Osnovne komponente APK datoteke, [21]

Sa slike 12 vidljivo je kako osnovne komponente svake APK datoteke čine:

- **AndroidManifest.xml** datoteka,
- **META/INF**, koja se sastoji od MANIFEST.MF datoteke te obuhvaća meta podatke o dozvoli za JAR (engl. *Java ARchive*)[3] te se ovdje nalazi potpis apk datoteke,
- **classes.dex**, koji predstavlja Dalvik *bytecode* za primjenu u DEX formatu datoteke, odnosno Java (Kotlin) kod koji će aplikacija pokrenuti prema zadanim postavkama,
- **lib/**, koji čine izvorne knjižnice za aplikaciju i direktoriji za središnju procesorsku jedinicu (engl. *Central Processing Unit* – CPU), te
- **assets/**, gdje su pohranjene sve ostale datoteke za koje je moguće da će aplikacija trebati, [22].

AndroidManifest.xml ključna je komponenta koju mora imati svaka apk datoteka s točno tim nazivom, a sadrži bitne podatke o aplikaciji te je, između ostalog, prema [18], potrebna za deklariranje slijedećih komponenata:

[3] JAR (engl. *Java ARchive*): format datoteke neovisan o platformi koja omogućuje da se datoteka kompromitira i poveže sa Java aplikacijom, *applet*-om ili *WebStar* aplikacijom

- Naziva paketa aplikacije, koji se obično poklapa s nazivom koda, pri čemu alat za izgradnju sustava Android koristi naziv paketa aplikacije za određivanje lokacije entiteta kodova prilikom izrade projekta. Prilikom pakiranja aplikacije, alati za izradu izmjenjuju ovu vrijednost ID-em aplikacije iz *Grandle* izrađujućih datoteka, koja se koristi kao jedinstveni identifikator aplikacije u sustavu i u Google Play trgovini,
- Komponenata aplikacije, koje uključuju sve aktivnosti, usluge, prijamnike odašiljanja i davatelje sadržaja, pri čemu svaka komponenta mora definirati osnovna svojstva kao što je naziv Kotlin ili Java klase,
- Dozvola, koje su potrebne aplikaciji za pristup zaštićenim dijelovima sustava ili drugim aplikacijama. Također sadrži sva dopuštenja koja moraju imati druge aplikacije ako žele pristupiti sadržaju iz ove aplikacije, te
- Hardverskih i softverskih značajki koje pojedina aplikacija zahtijeva, što utječe na to koji uređaji mogu instalirati aplikaciju s Google Play-a.

Android OS mora znati da određena komponenta aplikacije postoji kako bi je mogao pokrenuti, što se postiže na način da aplikacije deklariraju svoje komponente u datoteci AndroidManifest.xml, [20]. Na slici 13 prikazan je jednostavan primjer koda datoteke AndroidManifest.xml.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.lightcone.webviewdemo">

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
          <action android:name="android.intent.action.MAIN" />

          <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:name=".Webscreen" android:label="Web"> </activity>
  </application>
  <uses-permission android:name="android.permission.INTERNET" />
```

Slika 13. Primjer koda datoteke *AndroidManifest.xml*, [20]

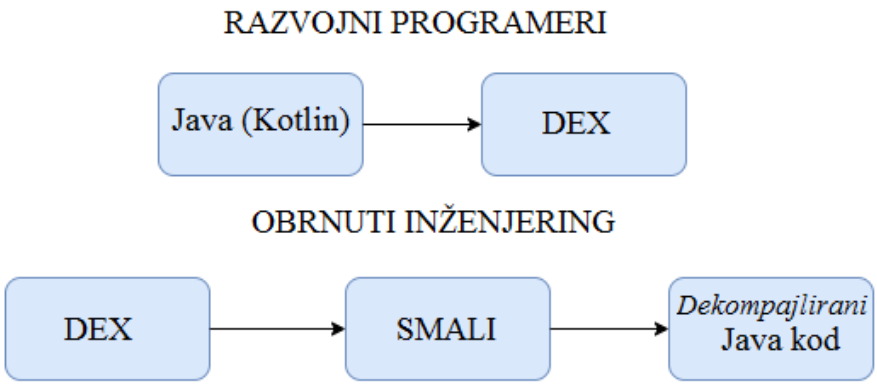
Sa slike 13 moguće je uočiti kako korijenski element datoteke AndroidManifest.xml čini atribut naziv paketa aplikacije, koji se obično podudara sa strukturom direktorija projekta. Prilikom izgradnje aplikacije u aplikacijski paket, alati za izgradnju koriste atribut naziva paketa u dvije svrhe: naziv se primjenjuje kao prostor za generiranu R.java klasu, koja se koristi za pristup resursima aplikacije te za rješavanje bilo kojih relativnih naziva klasa koja su deklarirana u datoteci manifest-a. Sa slike 7 također je vidljivo kako se unutar datoteke

AndroidManifest.xml nalazi lista dozvola za pristup osjetljivim korisničkim podacima ili značajkama sustava koje zahtjeva aplikacija, pri čemu se svaka dozvola prepoznaje jedinstvenom oznakom. [20].

3.5. Dalvik i Android Runtime virtualni stroj

Većina Android aplikacija pisana je u Java programskom jeziku, no Kotlin programski jezik također je podržan i interoperabilan s Javom. Umjesto da se Java kod pokreće pomoću Java virtualnog stroja (engl. *Java Virtual Machine* – JVM) poput desktop aplikacija, u Androidu se Java sastavlja u format *bytecode*-a Dalvik *Executable* (DEX). Za starije verzije Androida *bytecode* je preveo virtualni stroj Dalvik, dok se za novije verzije Androida koristi *Android Runtime* (ART), [22]. ART predstavlja upravljano vrijeme izvođenja koje koriste aplikacije i neke sistemske usluge, pri čemu su ART i njegov prethodnih Dalvik izvorno izrađeni za Android OS, [23].

Ukoliko programeri aplikacija pišu u Java programskom jeziku i kod se sastavlja u kod formata DEX *bytecode*, kako bi se proveo postupak obrnutog inženjeringa, proces se provodi u suprotnom smjeru, što je prikazano na slici 14.



Slika 14. Postupak izgradnje aplikacije i provođenja obrnutog inženjeringa, *[22]*

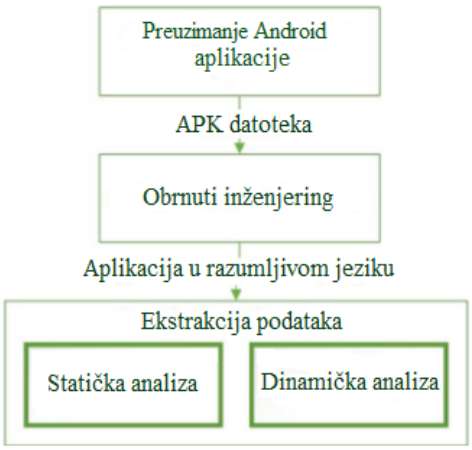
Sa slike 14 vidljivo je kako se prilikom provođenja obrnutog inženjeringa provodi postupak rastavljanja DEX *bytecode*-a u Smali te zatim dekompajlirani Java kod. Smali predstavlja čitljivu verzija *bytecode*-a Dalvik. Tehnički gledano, Smali i bakssmali su naziv alata (monter i rastavljač, respektivno), ali u Androidu često se koristi izraz "Smali" za upućivanje na upute, [22].

4. Obrnuti inženjering u svrhu analize sigurnosti Android aplikacija

Općenito govoreći, obrnuti inženjering definira se kao postupak kojim se gotov softver, odnosno izvršni kod analizira upotrebom specijaliziranih alata i postupaka, s ciljem proučavanja načina na koji on funkcionira, [24]. Shodno tome, obrnuti postupak je ekstrakcije znanja ili stvaranja nacrta iz nečega umjetno stvorenog uključujući rastavljanje i detaljnu analizu promatranih komponenata, te reprodukcije na temelju ekstrahiranih podataka. Općenito, obrnuti inženjering Android aplikacija provodi se u svrhu otkrivanja i razumijevanja načina na koji određena aplikacija funkcionira, njezine strukture i osnovnih karakteristika. Shodno navedenom, ciljevi provođenja postupka obrnutog inženjeringa određene Android aplikacije, prema [25] jesu:

- Pročitati te razumjeti programski kod,
- Pronaći sigurnosne propuste i osjetljive podatke unutar koda,
- Olakšati postupak prilagođavanja aplikacije novom hardveru, te
- Omogućiti izmjenu postojećeg koda, odnosno funkcionalnosti same aplikacije.

Pametni mobilni terminalni uređaji od lipnja 2004. godine postaju meta napadačima za umetanje zlonamjernih kodova unutar različitih aplikacija, čime se broj aplikacija koje u sebi sadrže neki oblik zlonamjernog softvera neprestano povećavao te su iz tog razloga uvedene različite mjere provjere sigurnosti aplikacija, [30]. Jednim od načina provjere sigurnosti Android aplikacija može se smatrati provođenje postupka obrnutog inženjeringa, u svrhu otkrivanja potencijalnih sigurnosnih propusta unutar koda te detekcije potencijalnog zlonamjernog softvera. Na slici 15 prikazan je postupak provođenja obrnutog inženjeringa Android aplikacija.



Slika 15. Postupak provođenja obrnutog inženjeringa, *[25]*

Sa slike 15 moguće je uočiti kako postupak provođenja obrnutog inženjeringa uključuje preuzimanje APK instalacijskog paketa, raspakiravanje APK datoteke i pretvorbu koda u čitljiv format te postupak ekstrakcije podataka koji je moguće provesti korištenjem statičke ili dinamičke analize te upotrebom odgovarajućih specijaliziranih alata.

Postupak obrnutog inženjeringa u osnovi se sastoji od dva temeljna procesa, rastavljanja (engl. *disassembly*) i dekompilacije (engl. *decompilation*). Rastavljanje podrazumijeva postupak kojim se jezik razumljiv stroju pretvara u tzv. *asssembly* jezik, odnosno programski jezik namijenjen programiranju računala, pri čemu je cilj postupka oblikovanje koda u format razumljiv i čitljiv čovjeku. S druge strane, dekompilacija predstavlja postupak obrnutog sastavljanja, odnosno pretvaranja izvršne datoteke u izvorni kod s čitljivim formatom. [25].

Prilikom razumijevanja postupka obrnutog inženjeringa, osnovno ga se može podijeliti u dvije dimenzije, prema [24]:

- Osnovne i napredne tehnike obrnutog inženjeringa, te
- Tehnike statičke i dinamičke analize.

Podjela obrnutog inženjeringa na osnovne i napredne tehnike te tehnike statičke i dinamičke analize neovisna je jedna o drugoj, odnosno osnovne tehnike obrnutog inženjeringa u pozadini automatski provode i statičku i dinamičku analizu, dok napredne tehnike također uključuju provođenje naprednih tehnika statičke i dinamičke analize, [24].

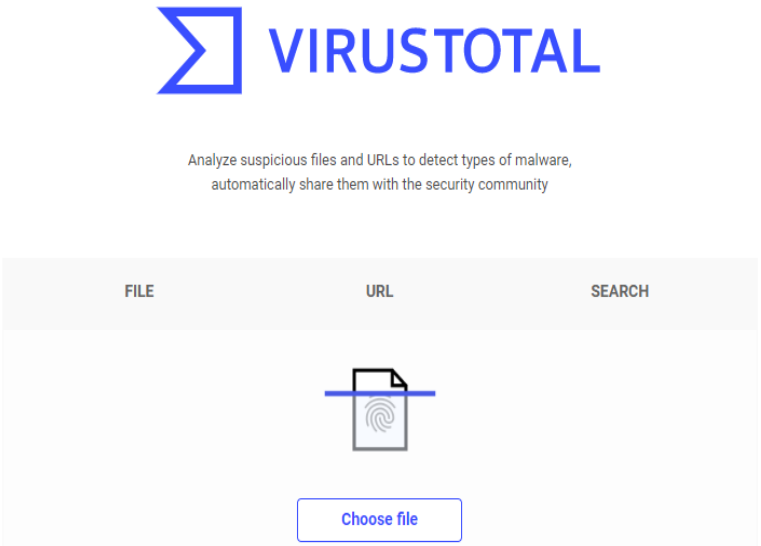
4.1. Osnovne i napredne tehnike obrnutog inženjeringa

Osnovne tehnike obrnutog inženjeringa podrazumijevaju automatizirani postupak provođenja obrnutog inženjeringa, pri čemu glavni dio posla provodi određeni specijalizirani program ili sustav te za njih nije potrebno posebno tehničko predznanje. Primjer osnovne tehnike provođenja procesa obrnutog inženjeringa jest korištenje sustava za automatsku analizu kao što je *VirusTotal* ili *HybridAnalysis*, [24].

VirusTotal predstavlja web stranicu koja krajnjem korisniku omogućava skeniranje potencijalno sumnjivih datoteka ili URL-ova, s ciljem potvrđivanja ili odbacivanja pretpostavke da su zlonamjerne. *VirusTotal* objedinjuje više od 70 antivirusnih alata različitih proizvođača među kojima su Avast, McAfee, Bitdefender, Kaspersky i sl., s ciljem poboljšanja sposobnosti detekcije zlonamjernog koda te koristi dodatne alate za statičku i dinamičku analizu datoteka i URL-ova, uzimajući u obzir komentare korisnika vezane za određenu datoteku ili URL. Za

analizu datoteka i URL-ova te identifikaciju i detekciju zlonamjernog softvera koriste se heurističke metode, prepoznavanje zlonamjernog potpisa te analiza metapodataka, [26].

VirusTotal dostupan je on-line putem linka *https://www.virustotal.com/*, a sučelje sustava prikazano je slikom 16.



Slika 16. Sučelje alata VirusTotal, [26]

Sa slike 16 vidljivo je kako *VirusTotal* nudi 3 opcije: učitavanje i skeniranje datoteke, skeniranje potencijalno sumnjive URL adrese, te mogućnost pretraživanja potencijalno sumnjive web adrese. *VirusTotal* prilikom učitavanja računa tzv. jedinstvenu *hash* vrijednost promatrane datoteke te je na taj način u mogućnosti povezati učitanu datoteku s istim takvim datotekama koje su učitali drugi korisnici, ili s datotekama koje se već nalaze u bazi antivirusnog alata, čime se ubrzava proces analize, [26].

S druge strane, napredne tehnike provođenja obrnutog inženjeringa podrazumijevaju ručno provođenje postupka od strane stručnjaka sa visokom razinom tehničkog znanja i razumijevanjem cijelog postupka. Napredne tehnike nužne su u slučaju pojavljivanja novog programa ili aplikacije, pri čemu sustavi kao što je *VirusTotal* ne daju puno korisnih informacija ili u slučaju postojanja zlonamjerne aplikacije ili programa, s ciljem preciznog otkrivanja načina na koji detektirana zlonamjerna aplikacija ili program radi, [26].

4.2. Tehnika statičke analize

Tehnika statičke analize koda podrazumijeva analizu softvera bez njegovog prethodnog pokretanja i izvršavanja, što uobičajeno podrazumijeva analizu strojnog, odnosno izvornog koda. Shodno tome, u većini slučajeva prilikom provođenja postupka statičke analize, analiza

se provodi na izvornom ili objektnom kodu, [24]. Prema [35] postoje 3 različita pristupa statičkoj analizi:

1. Pristup baziran na potpisu zlonamjernog koda,
2. Pristup baziran na dozvolama, te
3. Pristup baziran na analizi Dalvik *bytecode*-a.

Pristupom baziranim na potpisu zlonamjernog koda uobičajeno se koriste komercijalni antivirusni alati, pri čemu ova metoda izvlači semantičke podatke stvarajući jedinstveni potpis. Shodno tome, program se klasificira kao zlonamjerni ukoliko se potpis podudara s postojećim potpisom iz skupine zlonamjernih softvera. Nedostatak ovog pristupa jest u ograničenosti baze potpisa, budući da se ovom metodom mogu identificirati isključivo zlonamjerni programi koji su već klasificirani u skupini zlonamjernih softvera, dok zlonamjerni programi koji se ne nalaze u već postojećoj bazi ostaju neotkriveni, [35].

Druga navedena metoda temelji se na dozvolama koje određena Android aplikacija zahtijeva prilikom instalacije na uređaj. Prema zadanim postavkama, aplikacije nemaju dozvolu pristupa korisničkim podacima niti modifikacije sigurnosti sustava, pri čemu krajnji korisnik prilikom instalacije omogućava aplikaciji pristup resursima koje ona zahtijeva. Svaka Android aplikacija sadrži popis dozvola koje zahtijeva u datoteci *AndroidManifest.xml*, pri čemu je moguće da sva deklarirana dopuštenja nisu nužno potrebna za specifičnu aplikaciju. Shodno navedenom, pristup baziran na dozvolama temelji se na provjeri dozvola unutar *AndroidManifest.xml* datoteke, kako bi se detektirala eventualna neslaganja između dozvola koje bi aplikacija trebala zahtijevati s obzirom na njenu namjenu i svrhu, te dozvola koje aplikacija zapravo zahtijeva, [35].

Kao što je ranije objašnjeno, Android aplikacije pisane su u Java programskom jeziku, nakon čega se sastavljaju u Java *bytecode* te se nakon toga prevode u tzv. dalvik *bytecode*, odnosno .dex datoteku. Cilj pristupa temeljenog na analizi Dalvik *bytecode*-a jest prvenstveno pretvorba *bytecode*-a u jezik razumljiv čovjeku uz pomoć specijaliziranih alata te analiza ponašanja promatrane aplikacije, [35].

4.3. Tehnika dinamičke analize

Suprotno statičkoj analizi, tehnike dinamičke analize podrazumijevaju tehnike za čije je provođenje ključno pokretanje softvera, odnosno aplikacije unutar kontroliranog okruženja, (tzv. *sandbox*) te bilježenja što sve aplikacija tada radi; kojim datotekama aplikacija pristupa,

komunicira li mrežno s udaljenim poslužiteljima i sl., [24]. Prema [35] 3 su različita pristupa dinamičkoj analizi:

1. Pristup baziran na detekciji anomalija,
2. Analiza praćenjem osjetljivih informacija, te
3. Pristup baziran na imitaciji (engl. *emulation*).

Pristup baziran na anomalijama temelji se na praćenju ponašanja i aktivnosti uređaja za vrijeme izvođenja određene aplikacije i njezinih karakteristika, pri čemu praćenje ponašanja uređaja uključuje aktivnosti poput uspostave poziva, potrošnje baterije i sl. Rezultati praćenja potom se zapisuju unutar poslužiteljske baze podataka te se pomoću svojevrsnih algoritama za detekciju zlonamjernog ponašanja klasificiraju na sigurne ili zlonamjerne, [35].

Analiza praćenjem osjetljivih informacija, slično kao i prethodnom pristupu baziranom na detekciji anomalija, temelji se na praćenju izvora osjetljivih podataka kao što su podaci iz GPS sustava, kamere, mikrofona i sl. za vrijeme izvođenja aplikacije, kako bi se identificiralo potencijalno curenje podataka. Navedena metoda omogućuje učinkovito praćenje osjetljivih podataka, međutim nije u mogućnosti pratiti podatke koji napuštaju uređaj te se vraćaju u mrežnom odgovoru, [35].

Treći pristup dinamičke analize koda temelji se na imitaciji sustava pametnog terminalnog uređaja, što se obično provodi putem specijaliziranog programa, tzv. emulatora, koji je u mogućnosti imitirati OS na kojem se temelji promatrani mobilni uređaj. Pri tome se aplikacija instalira na emulator na isti način kao i na fizički mobilni uređaj te se prati njezino ponašanje bez da ona utječe na stvarne korisničke podatke. Takav sigurnosni mehanizam naziva se „pješčanik“ (engl. *sandbox*), pri čemu problem predstavlja to što je sve više zlonamjernih aplikacija „pametno“ te uspijeva prepoznati da ne radi o stvarnom fizičkom uređaju, pa iz tog razloga aplikacija neće pokrenuti zlonamjerni kod te isti neće biti detektiran, [35].

Kao i kod tehnike statičke analize, provođenje tehnike dinamičke analize moguće je korištenjem različitih alata namijenjenih za tu svrhu, kao što su MobSF, Droidbox, TaintDroid i sl.

5. Alati za provođenje statičke analize Java izvornog koda

Ručnu statičku analizu programskog koda Android aplikacija moguće je provesti korištenjem specijaliziranih alata kao što su dexdump, dex2jar, Dare, jd-gui, JadX-Gui, Jad i sl., a funkcionalnosti pojedinih alata opisane su kroz slijedeća poglavlja.

5.1. Alat za statičku analizu Java koda Dex2Jar

Svaka Android aplikacija izvršava se u vlastitoj instanci Dalvik virtualnog stroja, pri čemu Dalvik izvršava datoteke u .dex binarnom formatu. Pri tome, ključne komponente i programska logika Android aplikacije nalazi se unutar klase .dex datoteke, koju krajnji korisnik ne može vidjeti, te je iz tog razloga razvijen alat Dex2Jar za statičku analizu Java programskog koda. Shodno tome, glavna je svrha Dex2Jar alata pretvorba .dex datoteka u .class format, što čini kod razumljiv i čitljiv čovjeku te omogućuje pregled izvornog Java koda Android aplikacije, [30].

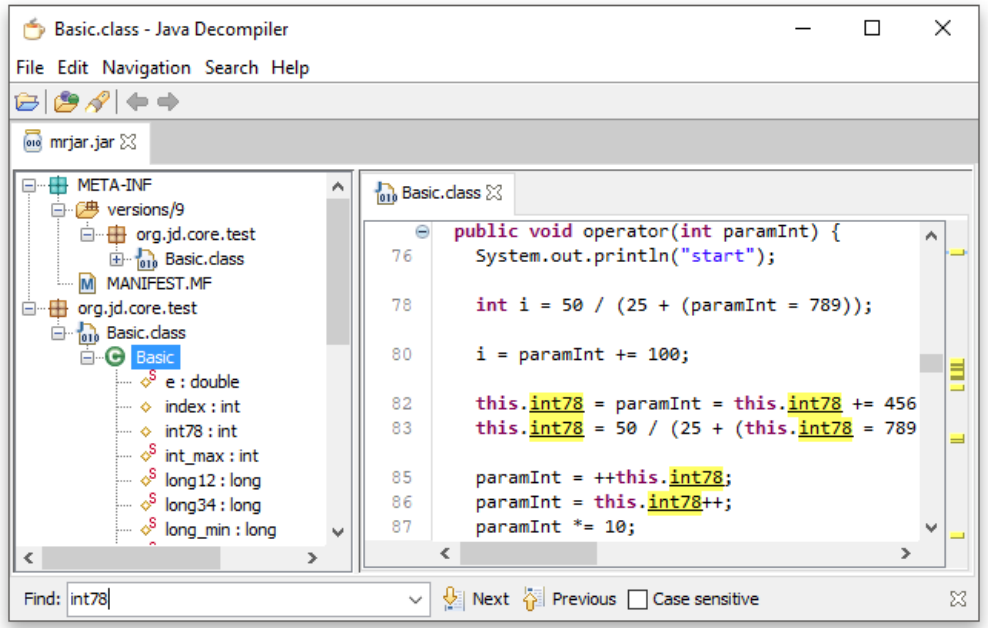
Alat za statičku analizu Java koda te Android aplikacija Dex2Jar besplatno je dostupan alat, a predstavlja pretvarač koji kao ulaz prima datoteku u .dex formatu pretvarajući ju u odgovarajuću .jar arhivu, s pripadajućim .class datotekama. Osnovna značajka Dex2Jar alata jest pretvorba classes.dex datoteke iz Android aplikacijskog paketa u pripadajuću classes.jar datoteku ili obrnuto. Pri tome je moguće pregledati izvorni kod analizirane Android aplikacije pomoću bilo kojeg Java dekompilera (engl. *Java decompiler*), pri čemu je izvorni kod aplikacije u potpunosti čitljiv. Korištenjem alata Dex2Jar također je moguće dobiti tzv. „*smali*“ datoteke izravno iz datoteke classes.dex i obrnuto, pri čemu je moguće izmijeniti izvorni kod aplikacije koja izravno radi s ovim formatom datoteke, [27]. Dex2Jar alat, prema [28], sadrži slijedeće komponente:

- **dex-čitač:** dizajniran je za čitanje datoteka u dex formatu, pri čemu ima jednostavno aplikacijsko programsko sučelje,
- **dex-prevoditelj:** dizajniran za posao pretvorbe, čitajući dex upute za dex-ir format, te pretvorbe u ASM format (engl. *Assembly Systems*) nakon provođenja optimizacije,
- **dex-ir**: koristi dex-prevoditelj te je dizajniran kako bi predstavljao upute za dex,
- **dex-alati**: koji rade s datotekama formata .class,
- **d2j-smali:** rastavlja dex datoteke na tzv. „smali“ datoteke te sastavlja dex datoteke iz „smali“ datoteka,

- **dex-pisač:** dizajniran na sličan način kao dex-čitač, a namijenjen je pisanju datoteka u dex formatu.

5.2. Alat za statičku analizu Java koda JD-Gui

JD-Gui samostalni je grafički uslužni alat koji omogućava prikaz Java izvornih kodova izravno iz .class datoteka. Alat za statičku analizu JD-Gui omogućava pregled rekonstruiranog Java izvornog koda uz pomoć trenutnog pristupa metodama i poljima, [29], a grafičko sučelje JD-Gui alata prikazano je slikom 17.



Slika 17. Sučelje alata za statičku analizu JD-Gui

Nakon pokretanja JD-Gui alata za ručnu statičku analizu Java koda, otvara se prozor prikazan na slici 17. Učitavanjem odabrane datoteke u JD-Gui alat moguće je pristupiti komponentama apk datoteke, uključujući *AndroidManifest.xml* datoteku koja sadrži sve bitne podatke o promatranoj aplikaciji, kao što je naziv i popis dozvola koje aplikacija zahtijeva.

5.3. JadX-Gui alat za statičku analizu Java koda

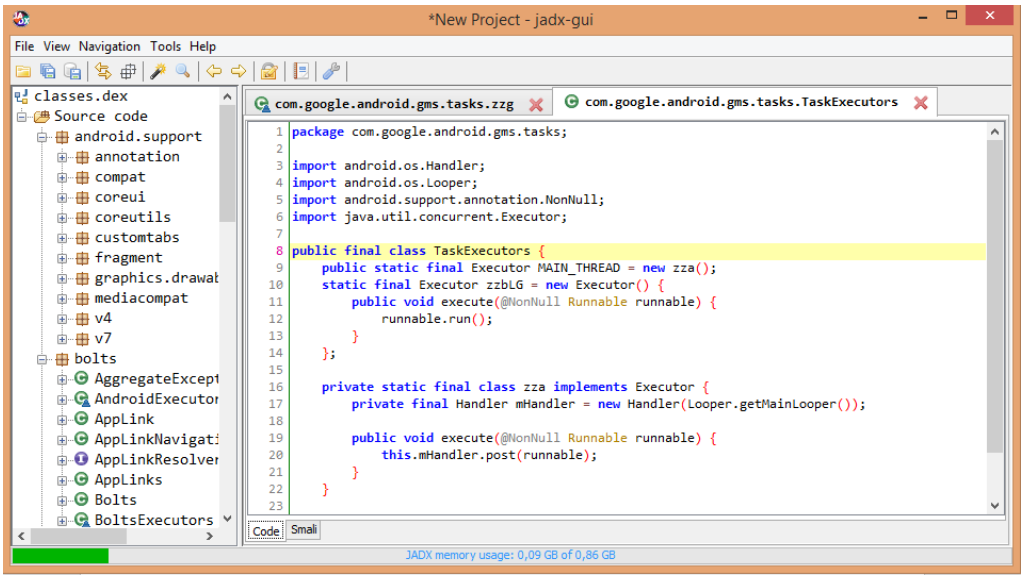
JadX-Gui alat može se smatrati jednim od najboljih sveobuhvatnih alata za provođenje ručne statičke analize Android aplikacija. Pri tome, JadX je besplatan softver otvorenog koda (engl. *free and open source softver*) koji je u mogućnosti rekonstruirati izvorni kod, dekodirati bitne podatke i metapodatke iz datoteke Android aplikacije u APK formatu, te prikazati te podatke krajnjem korisniku kroz pregledno grafičko sučelje. JadX alat dostupan je za operativne sustave Windows, macOS i Linux te ga je moguće preuzeti potpuno besplatno, pri

čemu nije potrebno provesti poseban postupak instalacije, već je isključivo potrebno raspakirati preuzetu arhivu, [24].

JadX-Gui alat je namijenjen provođenju statičke analize Java izvornog koda, rekonstruirajući Java kod iz datoteka formata Android .dex te APK datoteka. Neke od glavnih karakteristika JadX-Gui alata, prema [30] su:

- Pretvorba Dalvik *bytecode*-a u Java klase iz APK datoteka, te datoteka dex, aar i zip formata, što omogućuje pregled dekompiliranog koda s istaknutom sintaksom te
- Dekodiranje AndroidManifest.xml datoteke te drugih izvora iz *resources.arsc* datoteke.

Na slici 18 prikazano je grafičko sučelje alata za statičku analizu JadX-Gui.



Slika 18. Grafičko sučelje JadX-Gui alata za statičku analizu Java koda

JadX-Gui alat pokreće se otvaranjem direktorija *bin* iz preuzete arhive, te pokretanjem odgovarajuće datoteke „*jadx-gui.bat*“ na operativnom sustavu Windows, odnosno datoteke „*jadx-gui*“ na operativnim sustavima MacOS i Linux, [24]. Nakon pokretanja JadX-Gui alata potrebno je izabrati datoteku koju se želi analizirati, pri čemu datoteka mora biti u apk, dex, aar ili zip formatu. Nakon što je datoteka učitana u JadX-Gui alat, otvara se sučelje prikazano slikom 18, koje je podijeljeno u 2 dijela: s lijeve strane nalaze se komponente analizirane Android aplikacije hijerarhijski prikazane u obliku stabla, dok se s desne strane prikazuje izvorni kod odabrane komponente aplikacije, [24].

6. Praktični primjer obrnutog inženjeringa Android aplikacije

U svrhu izrade diplomskog rada proveden je postupak obrnutog inženjeringa nad dvije zlonamjerne Android aplikacije: WaTF-Bank koja predstavlja simulaciju aplikacije za mobilno bankarstvo, preuzete iz slobodno dostupne baze zlonamjernih programa za OS Android te aplikacije CovidLock, ucjenjivačkog softvera (engl. *ransomware*) zamaskiranog u aplikaciju namijenjenu praćenju i zaštiti od korona virusa u stvarnom vremenu. Obrnuti inženjering proveden je tehnikom statičke analize, upotrebom JadX alata.

6.1. Obrnuti inženjering u svrhu analize sigurnosti aplikacije CovidLock

Obzirom na veliku ekspanziju i razvoj tehnologije u vidu digitalizacije i informatizacije društva, velika većina djelatnosti odvija se elektroničkim putem, pri čemu krajnji korisnici terminalnih uređaja, zbog nepažnje ili nedovoljne educiranosti, svoje povjerljive podatke i informacije izlažu riziku. Nerijetko na taj način postaju žrtve tzv. *cyber* kriminala, pri čemu tzv. *cyber* kriminalci iskorištavaju situacije kada su korisnici najranjiviji, koristeći različite dramatične događaje koje uzrokuju određenu vrstu emocionalnog stresa i straha kod ljudi, kako bi ostvarili određeni profit ili korist. Iznimka nije ni pojava novog korona virusa[4], pri čemu su ubrzo nakon što su identificirani prvi slučajevi zaraze, istraživači tvrtke DomainTools primijetili neznatan porast korištenja domena koje u nazivu sadrže riječi korona virus ili Covid-19. Istraživački tim za sigurnost tvrtke DomainTools kontinuirano prati pojavu sumnjivih domena vezanih uz pojavu korona virusa te je na taj način detektirao sumnjivu domenu naziva *(coronavirusapp[.]site)*. Sa navedene stranice korisnik je bio u mogućnosti preuzeti aplikaciju namijenjenu praćenju pojave korona virusa u stvarnom vremenu, a koja zapravo u sebi sadrži ucjenjivački softver, tzv. *ransomware*, [38]. Ucjenjivački softver, pri tome, ima za cilj okoristiti se žrtvom na način da napadnutom krajnjem korisniku terminalni uređaj postane zaključan te time nedostupan za korištenje ili na način da kriptirani podaci postanu nedostupni krajnjem korisniku, odnosno vlasniku samog uređaja. Nakon inficiranja terminalnog uređaja nekim oblikom *ransomware*-a, od krajnjeg korisnika se najčešće traži neki oblik otkupnine u zamjenu za daljnje normalno korištenje uređaja, [39].

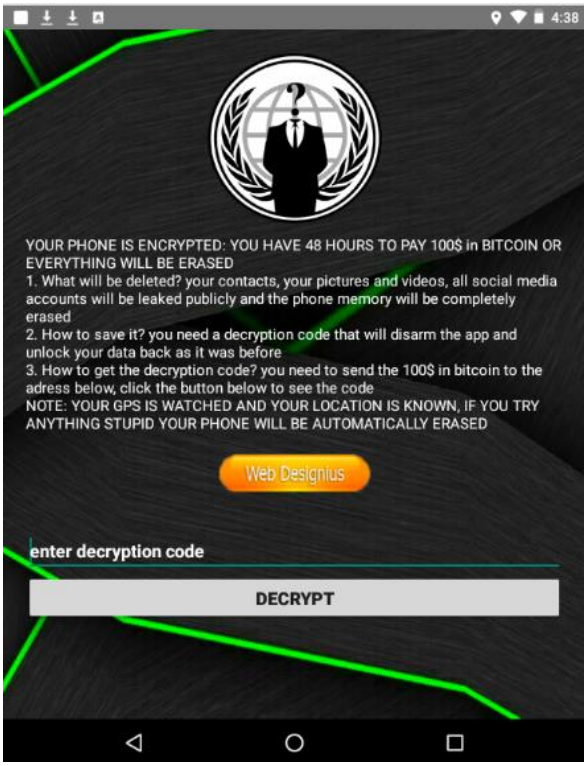
Budući da je aplikacija ubrzo detektirana kao ucjenjivački softver, koja od krajnjeg korisnika traži određeni svotu novca kako bi ponovo ostvario pristup svom terminalnom uređaju i podacima, stranica je u međuvremenu uklonjena, a SSL

[4] Korona virus (infekcija covid-19)

certifikati zlonamjerne domene *(coronavirusapp[.]site)* povezani su s drugom domenom naziva *(dating4sex[.]us)*, koja je također identificirana kao zlonamjerna aplikacija. Domena aplikacije prvotno je registrirana 8. ožujka 2020. godine, koristeći privatnost domene kako bi se prikrili detalji, pri čemu je domaćin *Wrathost*, pružatelj jeftinog zajedničkog servera. Iz tog se razloga domena nalazi na IP adresi koju dijeli s preko 100 drugih domena koje nisu međusobno povezane, [40].

Domena aplikacije CovidLock prvotno je sadržavala tzv. *iframe* oznaku[5] koja dolazi izravno sa *infection2020[.]com* stranice, web stranice neovisnog programera za praćenje vijesti vezane uz infekciju Covid-19 iz SAD-a, sa sjedištem u SAD-u. Nakon nekoliko dana web mjesto aplikacije CovidLock promijenjeno je u korištenje resursa DoMobile-a, pružatelja različitih legitimnih Android aplikacija, [40].

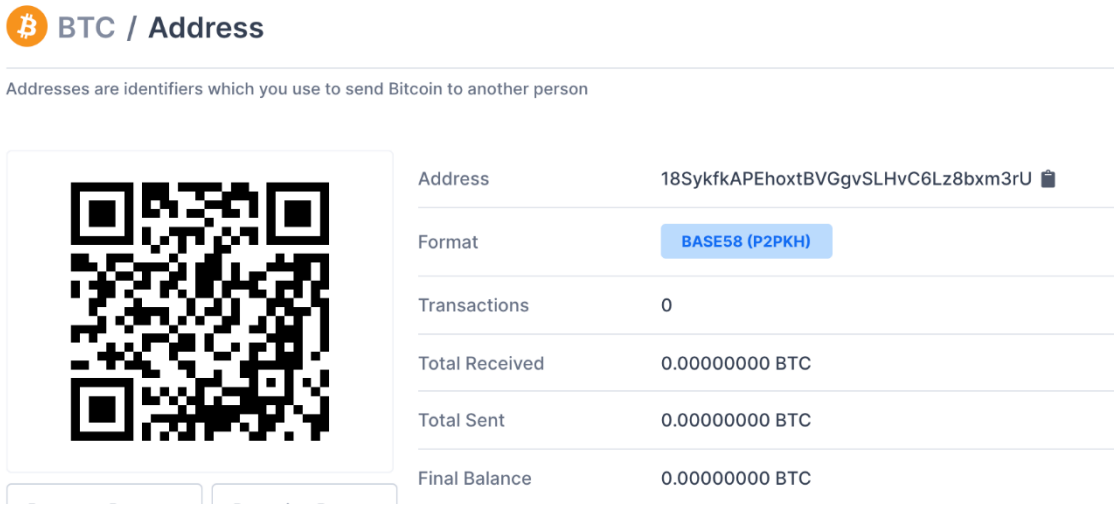
Preuzimanjem aplikacije CovidLock na terminalni uređaj, od krajnjeg korisnika zahtijeva se dozvola za administrativni pristup, te se potom otvara sučelje aplikacije koje od krajnjeg korisnika zahtijeva otkupninu za povrat njegovih podataka, što je prikazano slikom 19.



Slika 19. Početni zaslon aplikacije CovidLock, [40]

[5] Oznaka unutar HTML koda koja unutar sebe može sadržavati html dokument koji može i ne mora biti vezan za jednu web stranicu

Sa slike 19 vidljivo je kako aplikacija zahtijeva otkupninu od 100$ korištenjem bitcoin novčanika, u roku od 48h ili će svi podaci krajnjeg korisnika biti obrisani. Pri tome navodi da će svi kontakti, slike te video zapisi krajnjeg korisnika biti uklonjeni, te da će podaci javno procuriti na račune društvenih mreža. U slučaju da krajnji korisnik u navedenom vremenu uplati traženi iznos, navodi se kako će mu biti poslan enkripcijski ključ, s kojim će biti u mogućnosti otključati i ponovno pristupiti svojim podacima. Na slici 20 prikazan je Bitcoin novčanik ekstrahiran iz aplikacije CovidLock od strane tvrtke DomainTools.



Slika 20. Bitcoin novčanik ekstrahiran iz aplikacije CovidLock, [40]

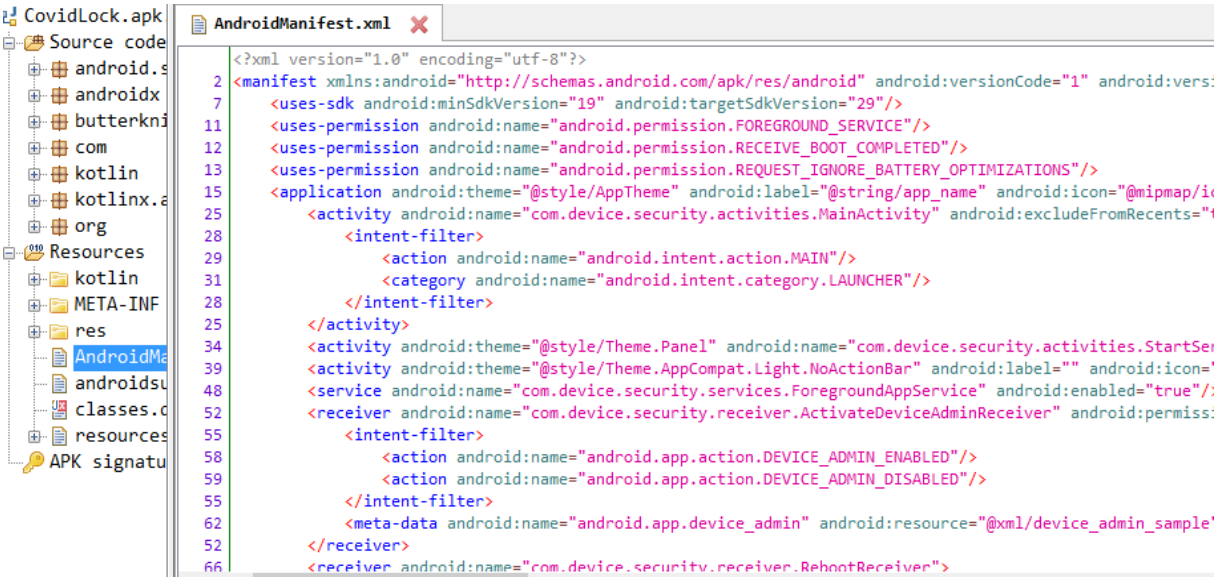
Gledajući ID bitcoin novčanika ekstrahiranog iz CovidLock aplikacije, do 15.03.2020. niti jedna žrtva nije platila otkupninu, što je vidljivo na slici 20.

Terminalni uređaji koji koriste *Nougat* verziju Android operativnog sustava zaštićeni su od ovakvog oblika napada, isključivo u slučaju da krajnji korisnik ima postavljenu lozinku za otključavanje početnog zaslona. Ukoliko krajnji korisnik nema postavljenu lozinku za otključavanje zaslona, i dalje je ranjiv te može postati žrtvom CovidLock ucjenjivačkog softvera, ukoliko zlonamjernu aplikaciju instalira na svoj terminalni uređaj, [38].

CovidLock novi je oblik ucjenjivačkog softvera koji koristi napad u obliku zaključanog zaslona terminalnog uređaja, kako bi napadač ostvario određenu novčanu korist. Ključno pitanje prije svakog početka provođenja postupka obrnutog inženjeringa, neovisno u koju se svrhu koristi, jest gdje započeti analizu izvornog koda, budući da Android aplikacije mogu biti iznimno velike te je gotovo nemoguće pregledati čitav kod. Najčešće se sa obrnutim inženjeringom započinje od pregleda interne datoteke *AndoridManifest.xml*, koja sadrži sve

bitne karakteristike promatrane aplikacije, te je prvi korak analiza dozvola koje promatrana aplikacija zahtijeva. Aplikacija CovidLock pisana je u Java programskom jeziku, a za početak statičke analize CovidLock apk datoteke, aplikacija je dekodirana pomoću alata JadX.

Dio koda datoteke *AndroidManifest.xml* i prikaz zahtijevanih dozvola aplikacije CovidLock prikazan je slikom 21.

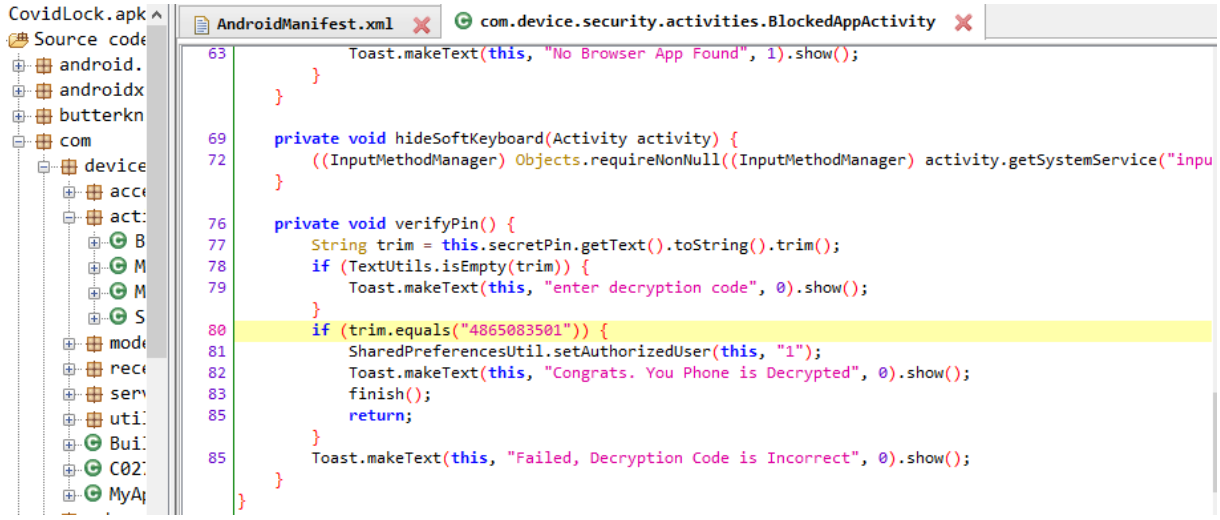


Slika 21. Prikaz djela koda datoteke AndroidManifest.xml aplikacije CovidLock

Sa slike 21 vidljivo je nekoliko bitnih karakteristika aplikacije CovidLock: prilikom prvog pokretanja, aplikacija od krajnjeg korisnika zahtijeva veliki broj dozvola, uključujući dopuštenje za rad s administrativnim povlasticama, dozvolu za rad u pozadini te pristup zaključanom zaslonu. Iako aplikacija tvrdi da zahtijeva pristup zaključanom zaslonu kako bi bila u mogućnosti obavijestiti krajnjeg korisnika da se pacijent zaražen korona virusom nalazi u njegovoj blizini, aplikacijama nije potreban pristup zaključanom zaslonu kako bi mogle slati obavijesti krajnjem korisniku. Način prikaza obavijesti kontrolira sam krajnji korisnik koji je u mogućnosti u postavkama svog uređaja postaviti koja će se vrsta obavijesti prikazivati u slučaju da je uređaj zaključan. Davanje dopuštenja za pristup zaključanom zaslonu napadaču omogućava rad s administrativnim pravima te mu omogućava upravljanje uređajem na sistemskoj razini.

JadX alat za provođenje statičke analize ima mogućnost unosa i pretraživanja teksta po određenoj ključnoj riječi, odnosno dijelova koda unutar aplikacije. Budući da CovidLock aplikacija zahtijeva od krajnjeg korisnika unos enkripcijskog ključa, koji će mu biti dostavljen nakon plaćanja traženog iznosa, može se zaključiti kako je isti zapisan unutar koda aplikacije.

Pretragom koda aplikacije na taj je način moguće pronaći potreban enkripcijski ključ, što je prikazano slikom 22.

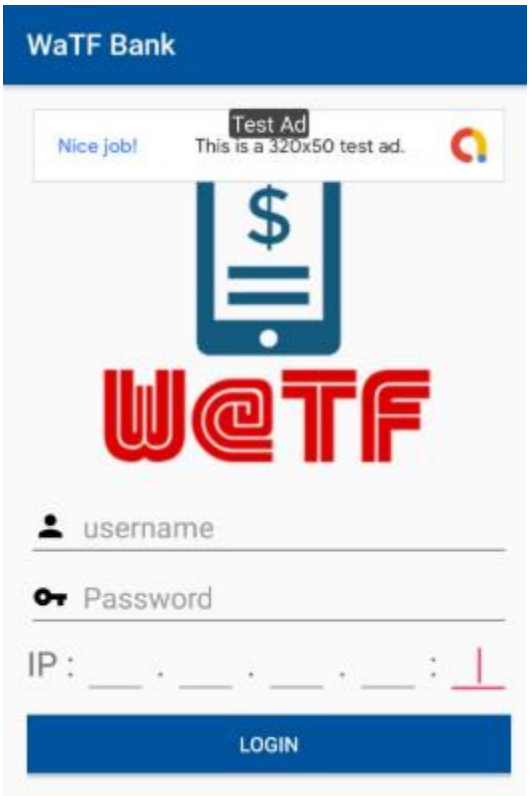


Slika 22. Enkripcijski ključ ucjenjivačkog softvera CovidLock

Sa slike 22 vidljivo je kako je enkripcijski ključ za otključavanje uređaja „4865083501“, što je vidljivo na liniji 80, te ukoliko krajnji korisnik unese navedeni ključ javiti će mu se poruka „*Congrats. You Phone is Decrypted.“* te će njegov uređaj biti otključan. U slučaju unosa pogrešnog enkripcijskog ključa krajnjem korisniku javlja se poruka „*Failed, Decryption Code is Incorrect“*, što je vidljivo na 85. liniji koda. U međuvremenu, sigurnosni stručnjaci tvrtke DomainTools ključ su objavili javno, kako bi svi koji su aplikaciju preuzeli, bili u mogućnosti vratiti svoje podatke.

6.2. Obrnuti inženjering u svrhu analize sigurnosti aplikacije WaTF-Bank

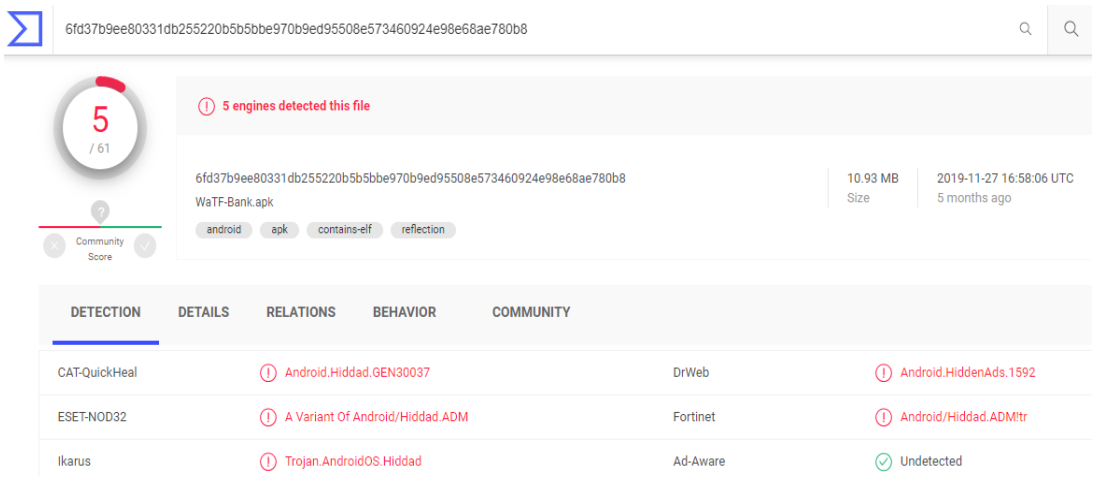
U svrhu izrade diplomskog rada te provođenja postupka obrnutog inženjeringa s ciljem analize sigurnosti Android aplikacija, sa službenih stranica je iz slobodno dostupne baze zlonamjernih programa preuzeta WaTF-Bank zlonamjerna aplikacija. WaTF-Bank (engl. *What-a-Terrible*-Failure) aplikacija je za mobilno bankarstvo, pisana u Javi, Swift4 te Objective-C programskim jezicima te Python-u kao pomoćnom poslužitelju, osmišljena kao testna aplikacija kako bi simulirala temeljne usluge mobilnog bankarstva, pri čemu sadrži preko 30 ranjivosti na temelju OWASP kategorizacije top 10 mobilnih rizika, a sučelje aplikacije prikazano je slikom 23.



Slika 23. Sučelje WaTF-Bank aplikacije

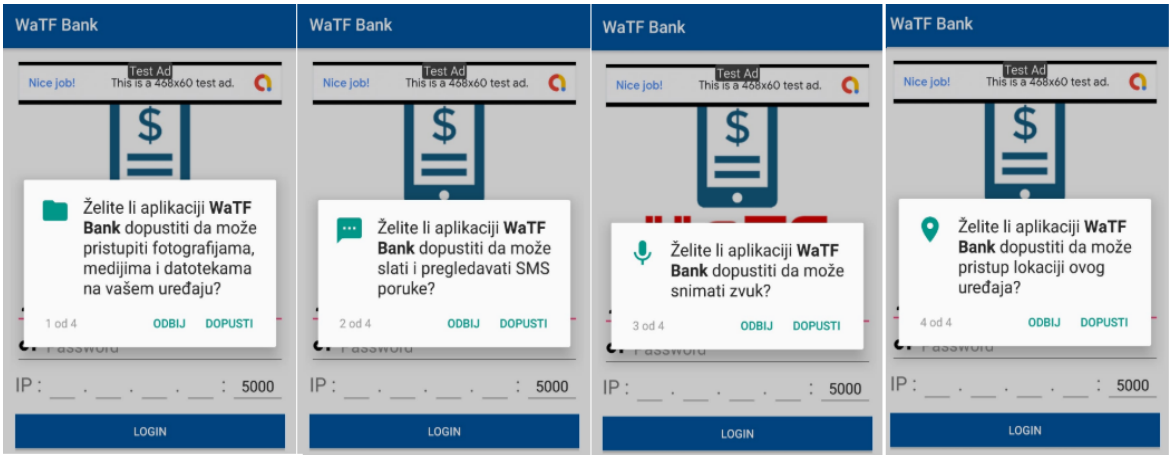
Aplikacija se predstavlja kao aplikacija za mobilno bankarstvo, pri čemu prilikom pokretanja izgleda kao originalna aplikacija mobilnog bankarstva te prikazuje polja za unos podataka potrebnih za prijavu: korisničko ime i lozinku, kao što je vidljivo na slici 23. Također, aplikacija zahtijeva unos IP adrese, što je također vidljivo na slici 23.

Prije provođenja postupka obrnutog inženjeringa ručnom statičkom analizom te upotrebom JadX alata, aplikacija je prvotno testirana kroz mrežni alat *VirusTotal*, kako bi se utvrdilo sadrži li aplikacija unutar sebe uistinu zlonamjeran kod. Rezultati skeniranja aplikacije alatom *VirusTotal* prikazani su slikom 24.



Slika 24. Rezultati skeniranja aplikacije WaTF-Bank korištenjem mrežnog alata VirusTotal

Sa slike 24 moguće je uočiti kako je unutar aplikacije WaTF-Bank mrežnim alatom *VirusTotal* otkriveno postojanje virusa poznatijeg pod imenom trojanski konj, čime se ova aplikacija može smatrati zlonamjernom. Prilikom preuzimanja WaTF-Bank aplikacije na uređaj te njezinog pokretanja, aplikacija zahtijeva niz dozvola od krajnjeg korisnika, što je prikazano slikom 25.



Slika 25. Prikaz zahtijevanih dozvola aplikacije WaTF-Bank

Sa slike 25 vidljivo je kako aplikacija zahtijeva pristup fotografijama, medijima i datotekama na uređaju, slanje i pregled SMS poruka, snimanje zvuka te pristup lokaciji korisnika.

Kao što je ranije navedeno, jedno od temeljnih pitanja prije provođenja postupka obrnutog inženjeringa, neovisno u koju se svrhu koristi jest gdje započeti analizu izvornog koda. Pri tome se najčešće sa obrnutim inženjeringom kreće od pregleda *AndoridManifest.xml* datoteke, koja sadrži sve bitne značajke promatrane aplikacije, te je prvi korak analiza dozvola koje promatrana aplikacija zahtijeva u odnosu na dozvole koje su potrebne kako bi se omogućila funkcionalnost same aplikacije. Pri tome je svaka zahtijevana dozvola kategorizirana odgovarajućom razinom rizika (engl. *permission level*). Razina kategorizira potencijalni rizik koji se podrazumijeva u sklopu pojedine dozvole te ukazuje na postupak koji bi sustav trebao slijediti prilikom određivanja dati aplikaciji pristup ili ne.

U tablici 1 prikazane su i objašnjene sve vrste razina osnovnih dozvola.

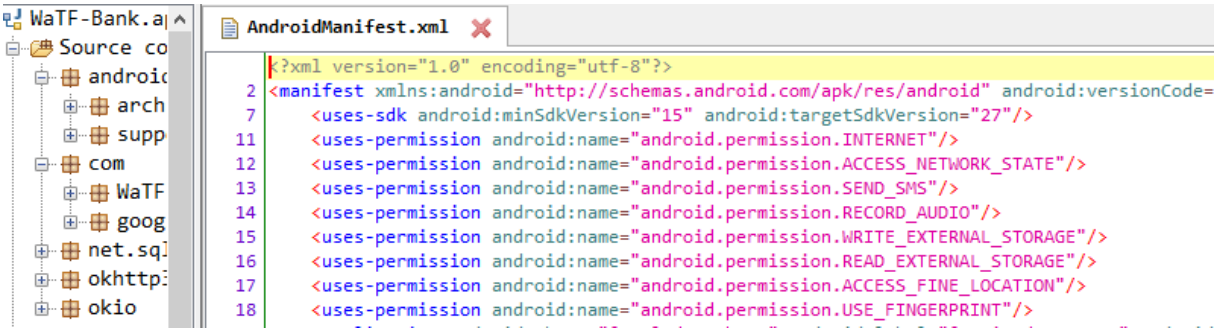
Tablica 1. Razine dozvola (engl. *permission level*), *[37]*

Razina dozvole	Značenje
"normal"	Zadana vrijednost. Dozvola nižeg rizika koja daje aplikaciji pristup izoliranim značajkama aplikacijske razine, s minimalnim rizikom za ostale aplikacije, sustav ili korisnika. Sustav automatski daje ovu vrstu dozvole aplikaciji koja ju zahtijeva, bez izričitog traženja dopuštenja od strane krajnjeg korisnika (iako je korisnik u mogućnosti pregledati dopuštenja prije instalacije)
"dangerous"	Dozvola visokog rizika koja daje aplikaciji pristup privatnim korisničkim podacima ili mogućnost kontrole uređaja što može imati negativan utjecaj na krajnjeg korisnika aplikacije. Budući da ova vrsta dozvole nosi potencijalni rizik, sustav neće odmah dati dozvolu, već će od krajnjeg korisnika zahtijevati potvrdu prije nastavka.
"signature"	Dozvola koju sustav dodjeljuje samo ukoliko je aplikacija potpisana s istim certifikatom kao i aplikacija koja je deklarirala tu dozvolu. Ukoliko se certifikati podudaraju, sustav automatski daje dozvolu bez obavještavanja korisnika ili traženja izričitog dopuštenja krajnjeg korisnika.
"signatureOrSystem"	Dopuštenje koje sustav dodjeljuje samo onim aplikacijama koje u posebnoj mapi sustava Android te su potpisane istim certifikatom kao i aplikacija koja je deklarirala tu dozvolu. Koristi se za određene posebne situacije u kojima više dobavljača imaju ugrađene programe u sliku sustava i trebaju eksplicitno dijeliti određene značajke jer se zajedno grade.

Iz tablice 1 vidljivo je kako postoje 4 različite razine dozvola: *normal*, *dangerous*, *signature* te *signatureOrSystem*. Svaka razina dozvole sadrži određenu razinu rizika za krajnjeg korisnika, što je detaljno objašnjeno u tablici.

Prilikom provođenja postupka obrnutog inženjeringa Android aplikacije WaTF-Bank korišten je alat JadX-Gui, pri čemu je sama aplikacija preuzeta u izvornom .apk formatu. Prvi

korak pri provođenju postupka obrnutog inženjeringa Android aplikacije jest provjera interne datoteke *AndroidManifest.xml* koja se nalazi unutar datoteke *resources*, a kod *AndroidManifest.xml* datoteke aplikacije WaTF-Bank, odnosno prikaz zahtijevanih dozvola aplikacije prikazan je slikom 26.



Slika 26. Sadržaj datoteke AndroidManifest.xml aplikacije WaTF-Bank (popis zahtijevanih dozvola)

Analiza koda *AndoridManifest.xml* datoteke započinje analizom dozvola koje zahtijeva promatrana aplikacija, te provjerom jesu li tražene dozvole u skladu s namjenom analizirane aplikacije. WaTF-Bank zlonamjerna je aplikacija, osmišljena kako bi simulirala temeljne usluge mobilnog bankarstva. Uobičajeno, aplikacije mobilnog bankarstva zahtijevaju autorizaciju od strane krajnjeg korisnika, unosom PIN-a, nakon čega se otvara glavno sučelje. Radi usporedbe, legitimna aplikacija mobilnog bankarstva mHPB, nakon autorizacije otvara sučelje koje prikazuje račune krajnjeg korisnika te stanje računa, a funkcije koje aplikacija omogućava su: uplata i prijenos novca na drugi račun, davanje naloga za podizanje e-gotovine, korištenje usluga e-kioska i slično.

Sa slike 26 moguće je uočiti kako se unutar datoteke *AndoridManifest.xml* nalazi popis slijedećih dozvola koje WaTF-Bank aplikacija zahtijeva od krajnjeg korisnika:

- **„android.permission.INTERNET“:** dozvola je koja omogućava aplikaciji pristup Internetu, pri čemu je razina dozvole „normal“, što znači da će prilikom instalacije WaTF-Bank aplikacije na uređaj, sustav prema zadanim postavkama dopustiti aplikaciji pristup Internetu, bez izričitog traženja dopuštenja od strane krajnjeg korisnika. Pri tome je u slučaju originalne i legitimne aplikacije za mobilno bankarstvo također potreban pristup Internetu, kako bi aplikacija mogla normalno funkcionirati.
- **„android.permission.ACCESS_NETWORK_STATE“**: slično prethodnoj, ova dozvola omogućava aplikaciji pristup podacima vezanim uz mrežu, pri čemu je

razina dozvole također „normal“, te sustav prema zadanim postavkama daje dopuštenje aplikaciji, bez potrebe za dopuštenjem samog krajnjeg korisnika.

- **„android.permission.SEND_SMS“**: dozvola je koja aplikaciji omogućava slanje SMS poruka, što nije funkcionalnost legitimne aplikacije mobilnog bankarstva. Razina ove dozvole pri tome se kategorizira se kao „dangerous“, budući da je ovo strogo ograničeno dopuštenje, [36].
- **„android.permission.RECORD_AUDIO“**: dozvola je koja aplikaciji omogućava snimanje zvučnih zapisa, pri čemu je razina dozvole kao i kod prethodne kategorizirana sa „dangerous“, što znači da može imati negativan utjecaj na krajnjeg korisnika aplikacije te nosi potencijalni rizik. Legitimna aplikacija mobilnog bankarstva ne zahtijeva ovu dozvolu, budući da snimanje zvučnih zapisa nije funkcionalnost takvog oblika aplikacije, čija je primarna funkcija omogućavanje bankovnih transakcija.
- **„android.permission.WRITE_EXTERNAL_STORAGE“**: svaka aplikacija kojoj krajnji korisnik da ovu dozvolu, implicitno daje dopuštenje i za dozvolu **„android.permission.READ_EXTERNAL_STORAGE“**. Ove dvije dozvole omogućavaju aplikaciji čitanje i pisanje po vanjskoj memoriji, a razina obje dozvole klasificira se kao „dangerous“ te dopuštanje ovih dozvola potencijalni je rizik za krajnjeg korisnika, budući da aplikacija mobilnog bankarstva prikuplja osjetljive podatke krajnjeg korisnika kao što je broj kartice, PIN brojevi, stanje računa i slično.
- **„andorid.permission.ACCESS_FINE_LOCATION“**: dozvola je koja omogućuje aplikaciji praćenje stvarne lokacije korisnika dok se aplikacija izvodi u pozadini sustava, pri čemu je razina klasificirana kao „dangerous“ te davanje ove dozvole predstavlja potencijalni rizik za krajnjeg korisnika, budući da njegova lokacija postaje poznata trećoj strani.
- **„android.permission.USE_FINGERPRINT“**: omogućuje aplikaciji prepoznavanje otiska prsta, pri čemu je razina klasificirana kao „normal“, što se može povezati sa legitimnom aplikacijom za mobilnom bankarstvo, budući da otisak prsta predstavlja način autorizacije i predstavlja pristupnu točku prilikom ulaska u sustav.

Iz navedenog moguće je zaključiti kako aplikacija WaTF-Bank sadrži neke sumnjive dozvole kao što su dozvola za slanje SMS poruka, pristup lokaciji krajnjeg korisnika te čitanje

i pisanje po vanjskoj memoriji, a koje se ne mogu povezati sa legitimnom aplikacijom mobilnog bankarstva te nisu nužne za njezino funkcioniranje.

Unutar *AndroidManifest.xml* datoteke također se nalazi popis aktivnosti aplikacije WaTF-Bank, što je prikazano slikom 27.

Slika 27. Aktivnosti aplikacije WaTF-Bank

Sa slike 27 vidljivo je kako aktivnosti aplikacije WaTF-Bank uključuju unos PIN-a, pri čemu sustav provjerava njegovu točnost te ukoliko je uneseni PIN ispravan, otvara se početna stranica aplikacije (engl. *Home*) koje potom nudi mogućnosti obavljanja bankovnih transakcija te pregled povijesti transakcija. Sa slike 27 također je moguće uočiti kako se unutar elementa filter namjere (engl. *intent-filter*) nalazi slijedeći kod:

<action android:name="com.WaTF.WaTFBank.SEND_SMS"/>

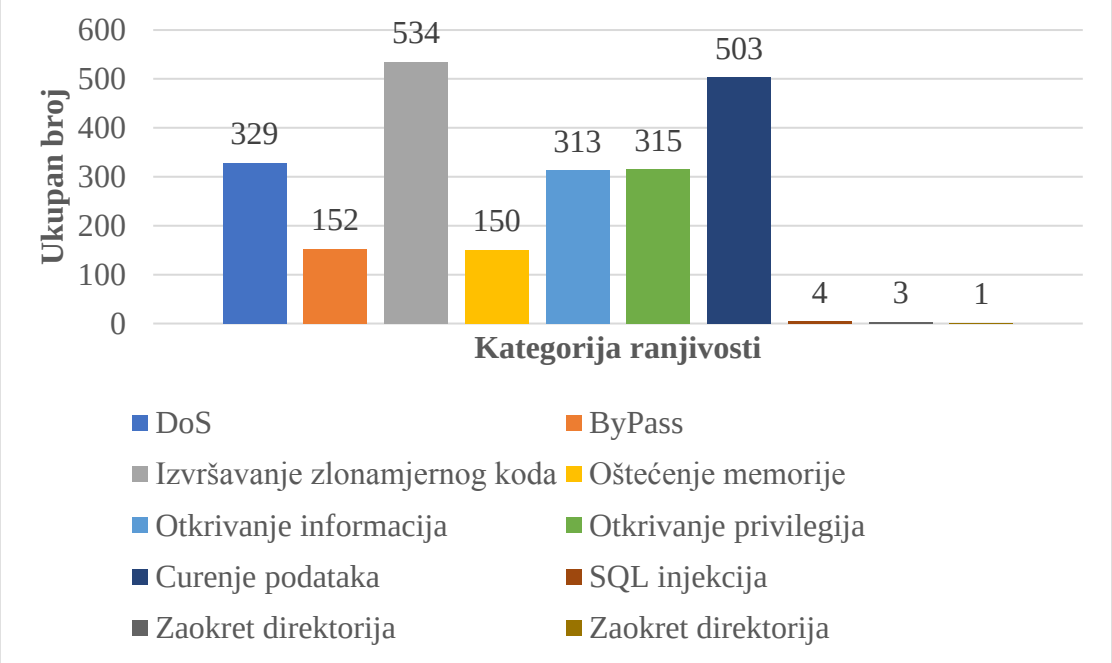
Filter namjere predstavljaju asinkronu komunikaciju kojoj je cilj omogućiti jednom objektu pozivanje radnji nekog drugog objekta. Element filter namjere nalazi se unutar elementa „*receiver*" koji predstavlja deklarirani prijamnik odašiljanja kao jednu od komponenata aplikacije, a omogućava aplikaciji primanje namjera koje emitira sustav ili druge aplikacije, čak i kada se druge komponente aplikacije ne pokreću, [38]. Iz navedenoga moguće je zaključiti kako je jedna od akcija koju aplikacija WaTF-Bank poduzima slanje SMS poruka, a bez znanja krajnjeg korisnika, čime se aplikacija može smatrati zlonamjernom, budući da slanje SMS poruka ne predstavlja funkcionalnost aplikacije koja se predstavlja kao aplikacija mobilnog bankarstva.

7. Zaštita aplikacija od obrnutog inženjeringa

Kao što je navedeno ranije u radu, Android aplikacije većinom su pisane u Java programskom jeziku, pri čemu je moguće otkriti kod aplikacije provođenjem postupka obrnutog inženjeringa te korištenjem specijaliziranih alata koji se koriste u tu svrhu. Više je razloga za provođenja postupka obrnutog inženjeringa, pri čemu su neki od njih:

- Ubacivanje zlonamjernog koda,
- Otkrivanje zlonamjernog koda,
- Identificiranje načina rada aplikacije,
- Otkrivanje osjetljivih informacija te
- Krađa koda.

Prema istraživanju provedenom od strane tvrtke Cvedetails (*The ultimate security vulnerability data source*) OS Android imao je ukupno 414 sigurnosnih ranjivosti detektiranih 2019. godine, a na grafikonu 2 prikazana je razdioba vrsta sigurnosnih ranjivosti operativnog sustava Android u razdoblju od 2009. do 2019. godine.



Graf 2. Razdioba kategorija sigurnosnih ranjivosti OS Android u razdoblju 2009-2019. godine, *[46]*

Sa grafikona 2 vidljivo je kako je najveći udio sigurnosnih ranjivosti operativnog sustava Android u razdoblju 2009.-2019. godine u vidu napada umetanjem i pokretanjem zlonamjernog koda unutar aplikacija, na koje otpada 21% ukupnog broja.

7.1. Načini zaštite Android aplikacija od postupka obrnutog inženjeringa

Uzevši u obzir razloge iz kojih se provodi postupak obrnutog inženjeringa, važno je znati zaštiti Android aplikaciju na ispravan način, naročito razvojnim programerima koji ne žele da njihov kod bude zloupotrijebljen u svrhe za koje on nije napravljen. Pri tome, aplikacija nikada ne može biti u potpunosti zaštićena od postupka obrnutog inženjeringa, ali se postupak svejedno može otežati na određene načine. Najčešći alat korišten u tu svrhu je ProGuard, koji predstavlja alat otvorenog koda pisan u Java programskom jeziku, koji štiti aplikacije od provođenja postupka obrnutog inženjeringa, primjenjujući metode kao što su:

- *Shrink* metoda – prepoznaje i uklanja neiskorištene klase, polja i atribute metoda,
- Optimizacija – analizira i optimizira kod različitim metodama, te
- Metoda zbunjivanja (engl. *Obfuscation*) – klasama, poljima i metodama se daju kratka i besmislena imena, [46].

Osim korištenja alata ProGuard, također postoje i drugi načini kako zaštititi izvorni kod aplikacije od obrnutog inženjeringa. Jedan od načina jest premještanje osjetljivog koda na web poslužitelje, što omogućuje kriptiranje koda na udaljenim poslužiteljima, a aplikacija koja ga koristi pristupati će mu putem mreže, prema potrebi. Nedostatak ovog vida zaštite jest ukoliko aplikaciju koristi veliki broj korisnika, potreban je ogroman klaster poslužitelja, što predstavlja veliki trošak te dugoročno to nije održivo rješenje. Nadalje, C/C++ kod teže je dekompajlirati radi čega veliki broj programera ključne kodove piše u .so datoteke. Kod se zatim lako može pretvoriti u tzv. *assembly* kod, ali bi postupak obrnutog inženjeringa u tom slučaju bio znatno otežan, [46].

Tijekom interakcije između poslužitelja i uređaja, programeri koriste SSL protokol za bolju sigurnost svog koda. Nedostatak jest postojanje nekoliko različitih metoda sadržanih u klasi implementiranoj unutar tzv. *SSLSocketFactory* sučelja, pri čemu metode prihvaćaju sve vrste certifikata, čineći aplikaciju ranjivom na napade čovjeka u sredini (engl. *man in the middle*). To može rezultirati gubitkom povjerljivosti podataka koji se prenose putem SSL/TSL protokola, pri čemu napadač može lako prekinuti vezu i dobiti vrijedne podatke jednostavnim pružanjem potvrde o vlastitom potpisu, [47]. Preostali načini zaštite aplikacija od obrnutog inženjeringa, prema [46] uključuju:

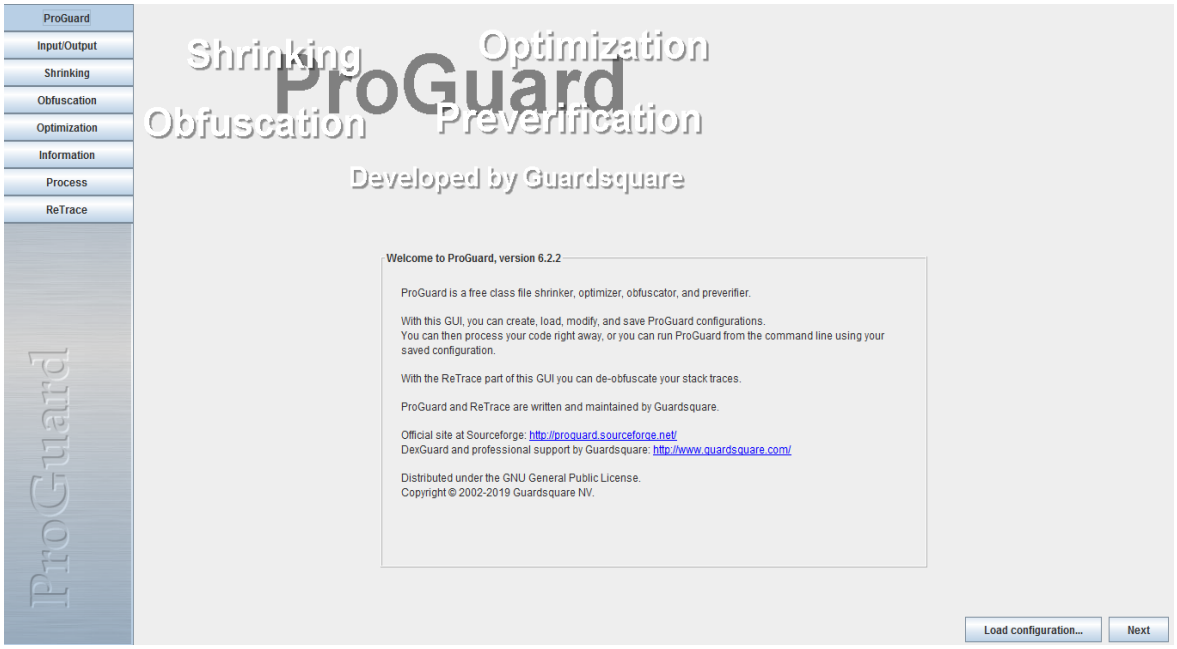
- Izbjegavanje upotrebe vanjske pohrane,
- Kriptiranje baza podataka,
- Upotreba hash algoritama,

- Sakrivanje API ključeva,
- Osiguravanje korisničkih vjerodajnica, te
- Izbjegavanje pohranjivanja vrijednosti u izvornom formatu.

Kako bi se aplikacije zaštitile od provođenja postupka obrnutog inženjeringa preporučljivo je osigurati korisničke vjerodajnice, pri čemu bi učestalost traženja vjerodajnica od krajnjeg korisnika trebala biti manja. Na taj način aplikacijama se omogućava izbjegavanje tzv. *phishing* napada. Također, preporučljivo je koristiti token autorizacije, pri čemu se korisnička imena i lozinke ne bi trebala pohranjivati na terminalni uređaj te je preporučljivo dovršiti početnu autorizaciju i korištenje kratkotrajnog tokena autorizacije. Za automatizaciju postupka provjere autentičnosti aplikacije, vlasnici zahtijevaju vjerodajnice korisnika te je u takvim slučajevima potrebno koristiti vjerodajnički objekt koji sadrži podatke za prijavu korisnika, [47].

7.2. Korištenje ProGuard alata u svrhu zaštite aplikacije od obrnutog inženjeringa

ProGuard predstavlja alat otvorenog koda, namijenjen zaštiti Android aplikacija od provođenja postupka obrnutog inženjeringa, koji otkriva i uklanja neiskorištene klase, polja, metode i atribute, optimizirajući time bajtkod aplikacije. Proguard alat moguće je koristiti na dva načina: u vidu naredbenog retka ili programa sa grafičkim sučeljem, pri čemu je grafičko sučelje prikazano slikom 28.

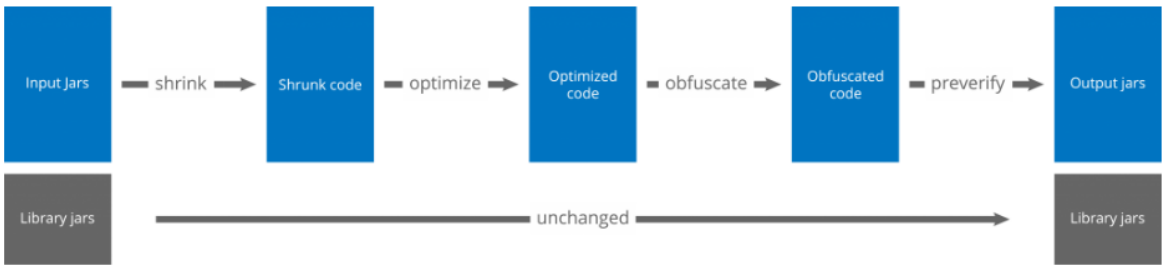


Slika 28. Grafičko sučelje alata ProGuard

Sa slike 28 vidljivo je kako alat ProGuard štiti Android aplikacije od postupka obrnutog inženjeringa primjenjujući tzv. *shrink* metodu, metodu optimizacije i zbunjivanja, pri čemu

korištenje navedenih metoda čine bazu manjom, učinkovitijom i težom za provođenje postupka obrnutog inženjeringa. Završni korak provjere (engl. *preverification*) dodaje informacije o potvrđivanju na klase, potrebne za Java Micro Edition te Java 6 i novije verzije, pri čemu svaki od navedenih koraka nije obavezan, [48].

Na slici 29 prikazan je postupak zaštite koda korištenjem alata ProGuard.



Slika 29. Koraci zaštite koda korištenjem alata ProGuard, [48]

Kao što je vidljivo na slici 29, ProGuard prvo čita ulazne podatke koji mogu biti u aar, war, ear, zip ili apk formatu ili u obliku direktorija te nakon čitanja ulaznih podataka provodi *shrink* metodu, metodu i te metodu zbunjivanja, pri čemu je opcionalno moguće dopustiti višestruke propusnice za optimizaciju. Nakon provođenja navedenih postupaka, Proguard bilježi rezultate u jedan ili više izlaznih podataka (engl. *output jars*). Ulazni podaci mogu sadržavati datoteke s resursima čija se imena i sadržaj mogu opcionalno ažurirati kako bi se održavala zakulisna imena klase. ProGuard zahtijeva specifikaciju tzv. *Library jars*, što u osnovi predstavlja biblioteke potrebne za sastavljanje koda, dok ih ProGuard koristi za rekonstrukciju ovisnosti klase koje su potrebne za pravilnu obradu, pri čemu same *Library jars* tokom cijelog postupka ostaju nepromijenjene, [48].

Kako bi zaštitili kod Android aplikacije korištenjem ProGuard alata potrebno je specificirati jednu ili više ulaznih točaka u kodu, kako bi se odredilo koji kod treba sačuvati te koji se može odbaciti. Pri tome su ulazne točke najčešće klase s glavnim metodama, programima, aktivnostima i sl. U koraku smanjivanja (engl. *shrink*) ProGuard polazi od tih točaka kako bi rekurzivno odredio koje se klase i članovi klase koriste, dok se svi drugi razredi i članovi klase odbacuju. U slijedećem koraku optimizacije koda ProGuard dodatno optimizira kod aplikacije, pri čemu se klase i metode koje nisu klasificirane kao ulazne točke mogu učiniti privatnim, statičnim ili konačnim, neiskorišteni parametri mogu se ukloniti, a određene metode mogu se istaknuti. U koraku zataškavanja ProGuard alat preimenovati će klase i članove klase koji nisu

klasificirani kao ulazne točke. U cijelom procesu, čuvanjem ulaznih točaka osigurava se mogućnost naknadnog pristupanja izvornim imenima, [48].

8. Zaključak

Eksponencijalni rast i razvoj tehnologije te informatizacija i digitalizacija društva pridonijeli su porastu broja aktivnih korisnika mobilnih terminalnih uređaja, a samim time doveli do porasta broja korisnika mobilnih aplikacija. Pri tome Android OS godinama zauzima većinski udio na tržištu te većina krajnjih korisnika posjeduje mobilni terminalni uređaj zasnovan na Android OS-u. Android aplikacije razvijane su većinom u Java programskom jeziku te su pohranjene u .apk formatu, pri čemu svaka apk datoteka sadrži sve ključne komponente nužne za pravilno izvođenje same aplikacije.

Mobilne aplikacije postale su epicentar trenutnih razvojnih trendova, dok se sigurnost Android OS temelji na mehanizmu baziranom na dozvolama. Pri tome, svaka Android aplikacija prilikom instaliranja na mobilni terminalni uređaj zahtjeva određeni set dozvola, koje omogućavaju ispravno funkcioniranje same aplikacije. Međutim, ukoliko je riječ o zlonamjernoj aplikaciji koja unutar sebe sadrži neki oblik zlonamjernog koda, moguće je da će ona zahtijevati veći set dozvola no što je potrebno za njezino nesmetano funkcioniranje. Prihvaćanjem tih dozvola, krajnji korisnik potencijalno sebe, svoj mobilni terminalni uređaj te svoje povjerljive podatke pohranjene na tom terminalnom uređaju izlaže opasnosti. Shodno tome, može se zaključiti kako će tzv. *cyber* kriminalci kontinuirano pronalaziti načine za ubacivanje i skrivanje zlonamjernog koda unutar izvornog koda aplikacije, međutim obrnuto inženjerstvo će gotovo uvijek pronaći način da to otkrije, samo je pitanje vremena kada.

U okviru diplomskog rada prikazano je postojanje velikog broja alata i tehnika namijenjenih analizi izvornog koda Android aplikacija, pri čemu se postupak analize izvornog koda naziva postupkom obrnutog inženjeringa te se može provoditi u različite svrhe. U svrhu izrade diplomskog rada, kao praktičan primjer, proveden je postupak obrnutog inženjeringa dviju Android aplikacija: WaTF Bank te aplikacije Covid Lock, a postupak obrnutog inženjeringa proveden je u svrhu analize sigurnosti Android aplikacija. Međutim, obrnuti inženjering također se može koristiti u svrhu ubacivanja zlonamjernog koda, izmijene te krađe koda aplikacije. Iz tog razloga, razvojnim programerima vrlo je važno zaštititi aplikaciju od potencijalnih izmjena koje mogu rezultirati negativnim ishodom te legitimnu aplikaciju pretvoriti u zlonamjernu. Najčešći alat korišten u svrhu zaštite izvornog koda aplikacije je ProGuard, koji predstavlja alat otvorenog koda, a temelji se na tri metode – *shrink* metodi, optimizaciji te metodi zbunjivanja. Pri tome, aplikacija nikada ne može u potpunosti biti zaštićena od provođenja postupka obrnutog inženjeringa, što se u jednu ruku može smatrati pozitivnom stranom, budući da

ukoliko se unutar aplikacije nalazi zlonamjerni kod, moguće ga je detektirati te na taj način zaštititi krajnjeg korisnika i njegov mobilni terminalni uređaj.

Iako je provođenje obrnutog inženjeringa kontinuirani postupak koji ide u korak s najnovijim postupcima ubacivanja zlonamjernog koda, na krajnjim korisnicima je da odgovorno koriste aplikacije kako bi izbjegli i prevenirali situacije u kojima bi im umetnuti kod mogao naštetiti.

Popis korištene literature:

[1] „Smartphone market - Smartphones - Statistics & Facts | Statista“, Preuzeto sa: https://www.statista.com/topics/840/smartphones/ (Zadnje pristupano: 02. travnja 2020.)

[2] „OSi terminalnih uređaja“, Autorizirana predavanja iz kolegija *Terminalni uređaji,* S. Husnjak

[3] „What is an API – Red Hat“, Preuzeto sa: https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces (Zadnje pristupano: 02. travnja 2020.)

[4] „Mobile Operating Systems - Statistics & Facts | Statista“, Preuzeto sa: https://www.statista.com/topics/3778/mobile-operating-systems/ (Zadnje pristupano: 03. travnja 2020.)

[5] Android Architecture – Tutlane, Preuzeto sa: https://www.tutlane.com/tutorial/android/android-architecture (Zadnje pristupano: 03. travnja 2020.)

[6] Singh K., „Mobile Phone Operating Systems: A Comparasion“, 2014., Dostupno: https://www.ijser.org/researchpaper/Mobile-Phone-Operating-Systems-A-Comparison.pdf (Zadnje pristupano: 03. travnja 2020.)

[7] „Components of mobile operating system“, Preuzeto sa: https://prezi.com/rzl_tvpeotkm/components-of-a-mobile-operating-system/ (Zadnje pristupano: 04. travnja 2020.)

[8] „Sistemas operativos móviles - Un murciano en el Polo“, Preuzeto sa: http://jpascu.blogspot.com/2011/03/sistemas-operativos-moviles.html (Zadnje pristupano: 04. travnja 2020.)

[9] „Beginning Android Application Development“, Wei-Meng Lee

[10] „Android – Statistics & Facts | Statista“, Preuzeto sa: https://www.statista.com/topics/876/android/ (Zadnje pristupano 05. travnja 2020.)

[11] „Croatia – StatCounter Global Stats“, Preuzeto sa: https://gs.statcounter.com/os-market-share/mobile/croatia (Zadnje pristupano: 06. travnja 2020.)

[12] „Android Architecture – AndroidClarified“, Preuzeto sa: https://androidclarified.com/android-architecture/ (Zadnje pristupano: 06. travnja 2020.)

[13] „Android – Architecture – Tutorialspoint“, Preuzeto sa: https://www.tutorialspoint.com/android/android_architecture.htm (Zadnje pristupano: 06. travnja 2020.)

[14] „Android Programming for Beginners“, J. Horton
[15] „Poglavlje 1 Android“, Prirodoslovno-matematički fakultet, Preuzeto sa: https://web.math.pmf.unizg.hr/~karaga/android/android_skripta.pdf (Zadnje pristupano: 07. travnja 2020.)
[16] „[AAR to DEX] Loading and Running Code at Runtime in Android Application“, A. Dangizyan, Preuzeto sa: https://medium.com/@artyomdangizyan/aar-to-dex-loading-and-running-code-at-runtime-in-android-application-69089a30c715 (Zadnje pristupano: 07. travnja 2020.)
[17] „Application Fundamentals | Android Developers“, Preuzeto sa: https://developer.android.com/guide/components/fundamentals (Zadnje pristupano: 08. travnja 2020.)
[18] „Android in Practice“, C. Collins, M. D. Galpin, M. Kaeppler, Dostupno: https://livebook.manning.com/book/android-in-practice/chapter-2/28 (Zadnje pristupano 08. travnja 2020.)
[19] „Android – Application Components – Tutorialspoint“, Preuzeto sa: https://www.tutorialspoint.com/android/android_application_components.htm (Zadnje pristupano 08. travnja 2020.)
[20] „Application Fundamentals“, Preuzeto sa: http://eagle.phys.utk.edu/guidry/android/applicationFundamentals.html (Zadnje pristupano 09. travnja 2020.)
[21] „Android Intervju Questions and Answers“, Preuzeto sa: https://www.andreasschrade.com/2017/02/23/android-interview-questions/ (Zadnje pristupano 10. travnja 2020.)
[22] „Android App Reverse Engineering 101 – GitHub Pages“, Preuzeto sa: https://maddiestone.github.io/AndroidAppRE/app_fundamentals.html (Zadnje pristupano 10. travnja 2020.)
[23] „Android Runtime (ART) and Dalvik“, Preuzeto sa: https://source.android.com/devices/tech/dalvik (Zadnje pristupano 11. travnja 2020.)
[24] „Reverzni inženjering Android aplikacija“, cert.hr, Preuzeto sa: https://www.cert.hr/wp-content/uploads/2020/01/Reverzni_inzenjering_Android_aplikacija.pdf (Zadnje pristupano: 18. travnja 2020.)
[25] „Reverse Engineering Android Applications“, Preuzeto sa: http://iisecurity.in/blog/344/ (Zadnje pristupano: 18. travnja 2020.)

[26] „Osnovna analiza zlonamjernog softvera pomoću online alata“, cert.hr, Preuzeto sa: https://www.cert.hr/wp-content/uploads/2019/07/analiza_zlonamjernog_softvera_online_alatima.pdf (Zadnje pristupano: 18. travnja 2020.)

[27] „Android Penetration Tools Walkthrough Series Dex2Jar, JD-GUI and Baksmali“, Preuzeto sa: https://resources.infosecinstitute.com/android-penetration-tools-walkthrough-series-dex2jar-jd-gui-baksmali/#gref (Zadnje pristupano: 19. travnja 2020.)

[28] „dex2jar Package Description“, Preuzeto sa: https://tools.kali.org/reverse-engineering/dex2jar (Zadnje pristupano: 19. travnja 2020.)

[29] „Java decompiler“, Preuzeto sa: http://java-decompiler.github.io/ (Zadnje pristupano: 22. travnja 2020.)

[30] „Reverse Engineering Android Applications“, Preuzeto sa: https://pentestlab.blog/2017/02/06/reverse-engineering-android-applications/ (Zadnje pristupano: 22. travnja 2020.)

[31] „Android Security Attacks and Defenses“, A. Dubey, A. Misra

[32] „Andorid Security Owerview“, Preuzeto sa: https://medium.com/@boshng95/android-security-overview-7386022ad55d (Zadnje pristupano: 25. travnja 2020.)

[33] „Android Security Model and Threat“, Preuzeto sa: https://hydrasky.com/mobile-security/android-security-model-and-threat/ (Zadnje pristupano: 25. travnja 2020.)

[34] „Emerging security threats for mobile platforms“, G. Delac, M. Silic, J. Krolo

[35] „Android Malware Detection & Protection: A Survey“, S. Arshad, A. Khan, M. A. Shah, M. Ahmed

[36] „MobileApp-Pentest-Cheatsheet“, Preuzeto sa: https://github.com/tanprathan/MobileApp-Pentest-Cheatsheet#reverse-engineering-and-static-analysis (Zadnje pristupano: 30 travnja 2020.)

[37] „<permission> | Android Developers“, Preuzeto sa: https://developer.android.com/guide/topics/manifest/permission-element (Zadnje pristupano: 30 travnja 2020.)

[38] „CovidLock: Mobile Coronavirus Tracking App Coughs Up Ransomware“, Preuzeto sa: https://www.domaintools.com/resources/blog/covidlock-mobile-coronavirus-tracking-app-coughs-up-ransomware (Zadnje pristupano: 10. svibnja 2020.)

[39] „Ransomware – CERT.hr“, Preuzeto sa: https://www.cert.hr/19795-2/ransomware/ (Zadnje pristupano: 10. svibnja 2020.)

[40] „CovidLock Update: Deeper Analysis of Coronavirus Android Ransomware“, Preuzeto sa: https://www.domaintools.com/resources/blog/covidlock-update-coronavirus-ransomware (Zadnje pristupano: 12. svibnja 2020.)

[41] Introduction to Activities | Android Developers, Preuzeto sa: https://developer.android.com/guide/components/activities/intro-activities (Zadnje pristupano 15. svibnja 2020.)

[42] Permissions Owerview | Android Developers, Preuzeto sa: https://developer.android.com/guide/topics/permissions/overview (Zadnje pristupano: 16. svibnja 2020.)

[43] Fragments | Android Developers, Preuzeto sa: https://developer.android.com/guide/components/fragments (Zadnje pristupano: 16. svibnja 2020.)

[44] Intent | Android Developers, Preuzeto sa: https://developer.android.com/reference/android/content/Intent (Zadnje pristupano: 20. svibnja 2020.)

[45] Avoid everse engineering of your Android app – Simform, Preuzeto sa: https://www.simform.com/how-to-avoid-reverse-engineering-of-your-android-app/ (Zadnje pristupano: 01. lipnja 2020.)

[46] Google Android: CVE Security vulnerabilities, Preuzeto sa: https://www.cvedetails.com/product/19997/Google-Android.html?vendor_id=1224 (Zadnje pristupano: 01. lipnja 2020.)

[47] How to prevent reverse engineering of your mobile apps?, Preuzeto sa: https://www.multidots.com/how-to-prevent-reverse-engineering-of-your-mobile-apps/ (Zadnje pristupano: 10. lipnja 2020.)

[48] ProGuard manual | Introduction | Guardsquare, Prezeto sa: https://www.guardsquare.com/en/products/proguard/manual/introduction (Zadnje pristupano: 10. lipnja 2020.)

[49] Google Play Store: number of apps 2019. | Statista, Preuzeto sa: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/ (Zadnje pristupano: 10. lipnja 2020.)

[50] Mobile Application Security Threats and Vulnerabilities 2019, Preuzeto sa: https://www.ptsecurity.com/ww-en/analytics/mobile-application-security-threats-and-vulnerabilities-2019/ (Zadnje pristupano: 10. lipnja 2020.)

[51] Mobile Application Architectures, Preuzeto sa: http://ptgmedia.pearsoncmg.com/images/0131172638/samplechapter/0131172638_ch03.pdf (Zadnje pristupano: 05. srpnja. 2020.)

[52] Android Hacker's Handbook, J. J. Drake, P. O. Fora, Z. Lanier, C. Mulliner, S. A. Ridley, G. Wicherski

[53] A Survey of Android Security Threats and Defenses, B. Rashidi, C. Fung

[54] Privilege Escalation Attacks on Android, L. Davi, A. Dmitrienko, A. Sadeghi, M. Winandy

Popis slika: